# AUTOMATING FORMAL SOFTWARE CONSTRUCTION

by

Gareth Carter


A Dissertation in Fulfillment

of the Requirements for the Degree

of

MASTER OF SCIENCE

at

NATIONAL UNIVERSITY OF IRELAND, MAYNOOTH

DEPARTMENT OF COMPUTER SCIENCE

July 2005

Head of Department: Professor Ronan Reilly

Principal Supervisor: Rosemary Monahan

# Declaration

I hereby certify that this material, which I now submit for assessment on the program of study leading to the award of M.Sc. in Computer Science, is entirely my own work and has not been taken from the work of others - save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: _____ Date: _____

# Abstract

This thesis aims to improve software quality by extending the support for formal software construction. By formal software construction is meant a process through which software is developed that is provably free from errors. In particular, the thesis provides a critical analysis of the software development tool Perfect Developer with respect to its support of the software construction process. In doing so, the thesis discusses the development of algebraic- and model-oriented specifications into object-oriented programs through refinement. These discussions result in the generation of a set of recommendations for an improved software construction tool.

# Acknowledgements

I would like to thank my mother, June, and my brother, David, for their support and encouragement during the preparation of this thesis. Without them I would have had many more difficulties and much less hair.

Many thanks go to my supervisor Rosemary Monahan for her continued patience and understanding. She always made herself available when I encountered dead ends and always furthered progress.

Special thanks go to Prof. Joseph M. Morris for his seemingly limitless knowledge. His guidance with the presentation of this thesis was invaluable and I daresay I would not have learned as much as I did without his input.

Finally, I would like to thank my wife, Liz Riley, for her love and comfort during this time.

Enterprise Ireland provided funding under the Basic Research Grant SC/03/278.

# Dedication

For James Carter,

my father.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

This thesis critically analyses the Perfect Developer software construction tool and its contribution to the development of object-oriented software. In particular it aims to improve tool support for the formal construction of object-oriented software. *Formal methods* enable the construction of software that is provably free from errors but often require high expertise for successful application. Support tools that simplify the application of formal methods and support formal software construction may be developed. A set of recommendations for a new support will be generated from an analysis of existing tools, in particular, Perfect Developer.

The analysis and recommendations are with respect to the support for *specification*, *implementation* and *verification* of software. Specifications, which define the essence of software under construction, must be highly expressive while remaining clean and compact. Implementations, which define the runtime behaviour of software precisely, must be efficient while exhibiting rich behaviour. Verification, which ensures software correctness, must be integrated into the development process while being accessible to all levels of expertise.

Perfect Developer is explored in depth as it is one of the most ambitious software constructions tools currently available. It supports the formal development of object-oriented software by refinement, and provides a formal verification of the code. It is built around a single language, Perfect, that supports both the specification and the implementation of software. Case studies and experiments are presented to illustrate software development with Perfect Developer and to support a critical analysis of the tool. The result of this analysis is the generation of a suite of recommendations for an improved tool that supports the construction of object-oriented software.

## 1.1 Supporting Specification

Specifications state the meaning of software without requiring procedural descriptions of how this behaviour is achieved. *Specification languages* are used to define specifications and may be *formal* or *informal*. Informal languages may hinder software development because they provide no means to ensure specifications are *complete*, *correct* and *consistent*. Formal languages allow the construction of more rigorous specifications that may be tested to ensure their quality. Formal specifications also guarantee a single interpretation provided the specification language has a clear semantics.

The cost of employing a formalism is often regarded as too high to merit the development of a good specification in advance of its implementation. Use of a formalism could be promoted by providing a specification language that is rich and expressive but still associated with popular programming languages. This association may be provided by making the specification executable (also referred to as *animation*) [44] and through *refinement* (of data and algorithms) [12]. The specification language

should offer developers a gentle learning curve inhabiting a single theory of software development. It is with respect to these qualities that we will evaluate specification languages presented in this thesis.

## 1.2 Supporting Implementation

Implementations define the procedural behaviour of software. There are a multitude of implementation languages available to developers today, catalogued by the *programming paradigm* they advocate. The functional programming paradigm [22] subscribes to an elegant, declarative style of implementation whereas the procedural programming paradigm [48] prefers a low-level, operational style. The *object-oriented* programming paradigm [73], which supports abstract software components be developed independently and later composed, is of interest in this thesis. It has been shown to give rise to an improvement in software quality in the long term [87] and is the most popular software development paradigm in use today.

Several theories of what it means for a language to be "object-oriented" have resulted from the varied and plentiful research [23, 6, 84]. The result is that the paradigm lacks a single mathematical foundation [51]. Support for software re-use, potentially offering platform or even programming language independence should be encouraged. The areas under dispute within the paradigm must be treated in a unified and *type safe* way. The executable programs generated from the implementation must be efficient. This thesis will keep these issues at the forefront of any analysis of implementation languages.

## 1.3 Supporting Verification

The verification process, which ensures the correctness of software, may be carried out by manual or automatic means. Verification of software, regarded as the great challenge of software development [21], has become a possibility with recent technological advances. Verification tools such as *symbolic executors*, *model checkers* and *theorem provers* have grown in scale and power. They can be used for *horizontal verification* of specifications to ensure consistency between specifications and for *vertical verification* of implementations to ensure their correctness with respect to the specification. *Validation* is the process by which completeness is proved. However, *validation* is an open research issued and beyond the scope of this thesis.

With the notable exception of the safety-critical community, verification tools have had a limited impact on the software development community. To be successfully applied these tools often require immense mathematical expertise. Support can be provided to the developer to make verification more accessible by automating elements of the verification process. This automation, however, should not have an adverse effect on the verification process. Verification output should be presented clearly to assist developers who need to study it. Common proof obligations should not require excessive assistance from the developer during their verification. These concerns will be used in assessing the verification support provided by any tool.

## 1.4 Perfect Developer: An Overview

Perfect Developer [34] is comprised of four components: the project manager; the Perfect language; the verifier; and the compiler. The project manager is an application

that is in charge of source code management and controls interactions between the other components. The Perfect language is an object-oriented specification language with an implementable subset. The verifier is a custom-built theorem prover that collects and attempts to discharge proof obligations for the software it is presented with. The compiler accepts code written in the implementable subset of Perfect and compiles it into Java, C++ or Ada95 code.

Perfect Developer can be run on most modern PC's with a Pentium compatible processor and at least 256MB of memory. The application will run under Windows XP, Windows 2000, Windows NT4 or Linux. For the purpose of this thesis, Perfect Developer Version 2.10 was installed on a Pentium 4 PC running Windows XP. All standard settings were used in the configurations except that Java was chosen as the implementation language. Since the work carried out in this thesis, a new version of Perfect Developer (Version 3.0) has been released which includes some changes to the verifier and some minor language features. A full list of these changes can be found at [1].

The tool has been used for teaching formal methods at Griffith College Dublin and at the National University of Ireland, Maynooth. Escher Technologies report an industrial use on a project of several hundred thousand lines of source code. Perfect Developer is used to illustrate the cutting edge in formal software development tools as it supports software specification, refinement and full verification.

## 1.5 Aims of the Thesis

The primary aim of this thesis is to provide an in depth analysis of the Perfefct Developer tool and its contribution to the development of object-oriented software.

The objective is to generate a set of recommendations that address the limitations faced by existing software development tools and hence to solve existing problems in the software development community.

## 1.6 Structure of the Thesis

In this chapter, the domain of the thesis is introduced, indicating the layout, goals and terminology that will reappear throughout the work.

Chapter 2 of the thesis defines the relevant terminology associated with formal software development. The terms specification, implementation and verification are further explained to ensure a thorough understanding of the field. The chapter discusses four distinct granularities at which specification may be undertaken. It clarifies the object-oriented programming paradigm as used in this thesis. Four techniques for program verification are also presented. Finally, the Perfect Developer software tool and its four major components are presented.

In chapter 3, the Perfect Developer software tool is compared with a collection of similar support tools. The chapter serves to discuss a variety of automated support tools available to modern developers and assess their success in penetrating industry. It concludes with a survey of the tools used during this research.

Chapter 4 presents the Perfect language, an object-oriented specification language with an implementable subset identified as its programming language. The core theory of Perfect is presented while simultaneously describing the syntax. The chapter acts as a tutorial in Perfect providing the reader with a gentle introduction to the language. A small supporting example is employed to illustrate the presentation.

In chapter 5, two medium sized case studies, constructed with the support of

Perfect Developer, are described. The case studies illustrate a wealth of features associated with Perfect Developer, demonstrating the requirements the tool places on a developer.

Chapter 6, an optional chapter, presents a selection of small examples that illustrate experiences with Perfect Developer. Development of the examples was driven by the case studies that were presented in chapter 5. These examples should be read to observe how Perfect Developer handles software development challenges in isolation.

Chapter 7 presents an analysis of Perfect Developer with respect to how it supports specification, implementation and verification of software. The chapter discusses the limitations that were presented in Chapters 5 and 6.

In chapter 8, a set of recommendations for SCALE, an automated tool that supports formal software construction, is presented. The chapter builds upon Perfect Developer, making suggestions on how the limitations described in Chapter 7 could be overcome and what other enhancements could be made. The recommendations aim to be practical and useful for modern software development.

Chapter 9 concludes the thesis by reviewing and evaluating the work that was carried out. The chapter also points to avenues of future work that may arise out of this research.

## 1.7 Conclusion

In this chapter, the domain and general direction of this thesis has been outlined. The importance of supporting specification, implementation and verification of software has been shown alongside a brief discussion of how this support may be automated. The aims of the thesis were presented as well as the means by which we hope to

achieve them.

# Chapter 2

# Formal Software Development

In this chapter, the concepts at the core of this thesis are presented and defined. The specification, implementation and verification techniques that are in use today as part of formal software development are the focus of the chapter. Each technique is defined and illustrated by example when necessary. To complete the setting, Perfect Developer, the software development tool that is at the core of this thesis, is introduced.

## 2.1 Specification

A specification may be written at different granularities in order to represent distinct kinds of behaviours. Initially, the *requirements* of the software under construction are specified. These may be further refined into a *mathematical* specification that precisely captures the essence of the software, without constraining the final implementation. Reactive or multi-threaded software may require *temporal* specifications that define the software behaviour over time. It may be productive to embed *lightweight*

specifications with procedures that define their isolated behaviour to promote subsequent correct usage. In this section we overview these four specification techniques, describing their style, rigour and application in software construction. Analysis of specification language support will be made later with respect to these specification techniques.

### 2.1.1 Requirements Specification

The software development community has struggled to find an adequate language for the specification of requirements [100]. Natural or spoken language is commonly the first stage of requirements acquisition, but these specifications tend to allow multiple interpretations of the software [57]. Sometimes, a *graphical language* may be employed to include more rigour in the specifications. Examples of these languages are the Unified Modelling Language (UML) [9] and the Business Object Notation (BON) [81]. Unfortunately, this form of specification is often ambiguous and hence open to interpretation.

### 2.1.2 Mathematical Specification

By mathematical specification is meant the collection of specification techniques that rely upon mathematical notation to describe software at a very high level. In this thesis we are concerned with both *model-oriented specification* [41] and *algebraic specifications* [55]. Model-oriented specifications, supported by languages like VDM++ [38] and Object-Z [89], define the abstract data model of a system and the effect of the operations that act upon its state. Algebraic Specifications, supported by languages like CafeOBJ [77], define the behaviour of software by a set of axioms.

### 2.1.3 Temporal Specification

Temporal Specification encodes the elements of a software specification that occur over time or concurrently. Many languages that support temporal logic [18], the logic that describes temporal behaviour, have arisen. These include Communicating Sequential Processes (CSP) [50] and the Temporal Logic of Actions (TLA) [64]. These languages allow temporal operators like `always` or `eventually` to be declared.

### 2.1.4 Lightweight Specification

The term lightweight specification is used to denote specifications that do not define a complete software system, but rather local elements or procedures within the software. For example, *Design by Contract* [72] may be used in isolation with classes or methods of object-oriented software. The technique is based on Hoare logic [49], defining *pre-conditions* and *post-conditions* of methods as well as *class invariants*. The pre-condition of a method is the boolean term that defines the requirements that will ensure correct execution of the method. The post-condition of a method is the boolean term that defines the guarantees the method provide following successful termination. The invariant of a class is a boolean term that defines the state of the class at all observable points (i.e. preceding a method call or after a method terminates). Design by contract is supported by programming languages such as Eiffel [71] and C# [95].

## 2.2 Implementation

An implementation, or program, is an executable specification that fulfils the requirements of the software. Implementations are rarely generated automatically from

specifications. A refinement phase, that makes the specification more concrete, is usually necessary. The resulting specification can be further refined to an efficient executable piece of software. For the purpose of this work, implementations are treated as object-oriented programs that may be compiled. In this section we discuss the important elements of the implementation process and clarify the object-oriented programming paradigm as used throughout this thesis.

### 2.2.1 Refinement

Refinement[12] is a formal process that transforms one specification into another, preserving correctness. The final specification is delivered through a series of refinement steps that translate an abstract specification into a more concrete one. The abstract specification is the simplest representation of the system that defines the interface to the external world. The concrete implementation is the program that realizes this specification. Refinement offers a way to connect specification to implementation, but requires complex mathematics to be proved correct [36].

### 2.2.2 Object-Oriented Programming

A thorough description of the object-oriented paradigm may be found in [73], while the key terminology is presented here. The basic building block of the object-oriented programming paradigm is the **class**. A class defines the **features** of a set of run-time **objects**, thereby describing their type. The features of a class are the data **attributes** and the message passing **methods** that all the objects of this class possess. **Inheritance** allows classes be enhanced with new behaviour by constructing a new class, (the *child* class), from some existing class, (the *parent* class). During

inheritance, data attributes and methods may be introduced or hidden and methods may be **overridden**. **Polymophism** permits objects of the child class to masquerade their type as that of any *ancestor* class.

The object oriented paradigm as described above is not type secure and has the potential to allow a number of errors into apparently elegant software. The mechanism of overriding methods generally offers no guarantees that the new overriding method continues to implement the old. Polymorphism potentially results in software that behaves contrary to the developer's intention unless some guarantee is required. For example a method that determines the speed of a vehicle could be re-implemented to always return 5 miles per hour, resulting in incorrect behaviour when an object of the child class masquerades as its parent's type.

A similar problem is that of **binary methods**. A binary method is a method of a class that accepts a single parameter of the same type [24]. After inheritance, the binary methods of a class no longer hold their unique properties. See the section on *polymorphic perversity* in [73] for a clear account of the dangers when a male student masquerades as a neuter student allowing him to share a room with a female student. This problem is related to that of the **dynamic binding** of methods, wherein methods may not be bound correctly depending on the dynamic binding signature of the programming language [19].

However, the most troublesome element in object-oriented programming concerns the semantics of the programming language. Most object-oriented programming languages advocate **reference semantics**, whereby an attribute refers to an object in memory rather than actually evaluating to one. This semantics permit object *aliasing* to occur, whereby one object in memory is referenced by two distinct attributes

[103]. This is potentially hazardous to software as an object may change its state from one perspective, whereas from another it remains the same. The security of software that allows aliasing is extremely difficult to ensure and maintain [103]. The alternative to reference semantics is **value semantics** which avoids these issues by requiring all variables evaluate to a unique object. This provides less flexibility but greater security.

## 2.3 Verification

Verification ensures that software specification and implementation are consistent, correct and free from bugs. This proof may be obtained by hand, but due to the scale of modern software systems, it is preferable to employ tools that assist the verification. Examples of the technology behind these tools are *symbolic execution*, *theorem proving* and *model checking*. Symbolic execution reduces a procedure to the symbolic conditions on which the procedure relies and the symbolic guarantees it provides. Theorem Proving verifies whether or not an assertion follows logically from a set of axioms. Model Checking may validate a state based model against some safety property of the software. Verification of software is made possible by combining these technologies into a support tool. In this section, we discuss each technology in more detail.

### 2.3.1 Symbolic Execution

Symbolic execution [61] traces the path of a method using symbolic values to generate a symbolic value for the result while collecting verification conditions. The verification

conditions of a method are the collected pre-conditions of those methods called during the methods execution, as well as the final post-condition of the method. A program can undergo symbolic execution by modular execution of each method. It is essentially an adaptation of the Hoare approach but it employs a different verification mechanism more suited to tool-assisted verification.

Symbolic execution traces can be graphically represented as *symbolic execution trees* where the branching points correspond to `if` statements. Proof of `while` statements require the production of a *loop invariant* assertion, (i.e. that which is true at the start of execution and at the end of each iteration of the loop). In symbolic execution, while loops are explicitly annotated with invariant clauses that assist in proving the loop invariant. For total correctness a *loop variant* is required that defines the maximum number of iterations the loop requires. Therefore termination can be guaranteed by proving the variant always decreases but never below zero.

### 2.3.2 Theorem Proving

Theorem proving engines seek to reduce theorems to true by iteratively applying a selection of rules. The rules of the theorem prover are referred to as sequents and the result of applying a sequent is described by the sequent calculus [33] of the theorem prover. Theorem provers may be *automated* or *interactive*. Interactive theorem provers [39], or proof assistants, rely on human interaction in selecting the sequent that is to be applied, while automated theorem provers [20] use strategies that select the best sequent for application.

Automated theorem provers can be classified as rewriting or resolution theorem provers. Rewrite systems [52] transform the proof obligation to boolean values by

applying the axioms of its knowledge base. Resolution systems [85] iteratively unify theorems to generate new theorems that may resolve to the empty theorem, which is equal to true.

### 2.3.3 Model Checking

Model checking [46] is a completely automated reasoning technology that can prove software upholds certain safety properties. A model of the software is constructed that defines the states the software may enter, the conditions required for entering these states and the transitions between the states. This model checker accepts this model as input and performs a state space search, ensuring a safety property is true at all points. The checker exhausts all possible paths of execution before confirming or denying the safety property.

The major drawback to model checking technologies is the *State Space Explosion Problem*[4]. This problem refers to the exponential increase in potential paths of a model as the number of states increases. Model checking serves a useful, but limited, function in reasoning support.

## 2.4 Perfect Developer

Perfect Developer [34] is a software development tool comprising of four components: the project manager; the Perfect language; the verifier; and the compiler. The project manager is an application that is in charge of source code management and which controls interactions between the other components. The Perfect language is an object-oriented specification language with an implementable subset. The verifier is

a custom-built theorem prover that collects and attempts to discharge proof obligations for the software it is presented with. The compiler accepts code written in the implementable subset of Perfect and compiles it into Java, C++ or Ada95 code. In this section we present each of these components in more detail.

### 2.4.1 Project Manger

The project manager is the graphical user interface between Perfect Developer and its users (see Figure 2.1 and the accompanying Table 2.1). Its primary purpose is to manage project settings and to list the files included. Source files can be added and removed to a project, though their editing is performed by some external third party editor [59, 54]. Integration with UML modelling tools is also available through the *import xmi* feature. The verifier can be set to produce more or less detail in proofs, and the developer can increase or decrease processing time allotted to the verifier. The compiler can be customized to generate Java, C++ or Ada95 from Perfect source code.

### 2.4.2 Perfect Language

Perfect is an object-oriented specification language that has an implementable subset identified as its programming language. The language is both Z like, providing a library of useful collection and structure types, and Java like, in supporting encapsulation via classes, message passing and inheritance. Specifications in Perfect may take the form of *algebraic properties* (i.e. algebraic axioms) or design by contract assertions. A refinement step is promoted by allowing a class to have two levels of data attributes, one for specification and one for implementation. Perfect will be the

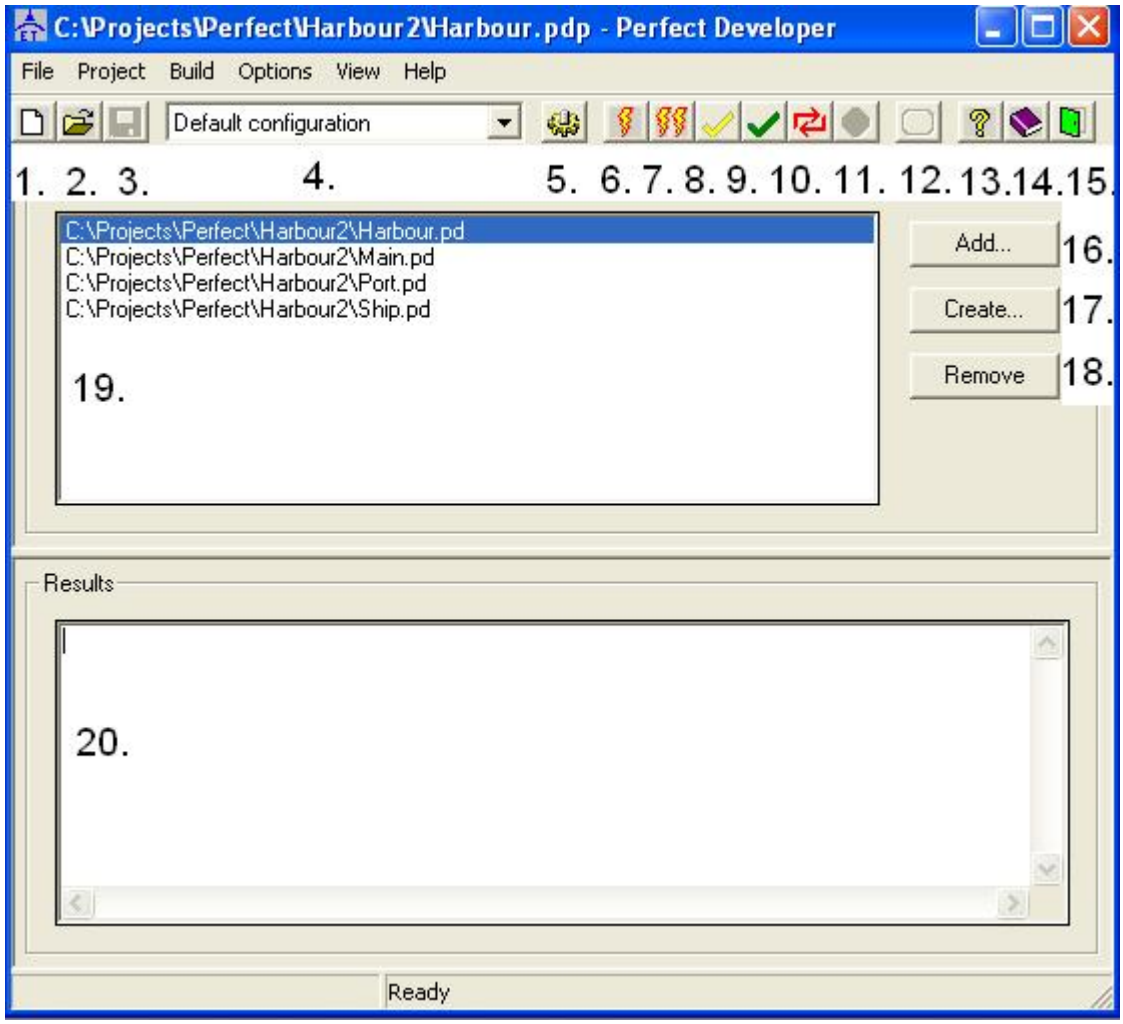


Figure 2.1: Perfect Developer GUI

| | | |
|---|---|---|
| 1 | New Project | Creates a new Project |
| 2 | Open Project | Opens an existing Project |
| 3 | Save Project | Saves current Project |
| 4 | Configuration Manager | Changes the project build configurations |
| 5 | Settings | Opens the Settings window |
| 6 | Build | Launches the build/compile task |
| 7 | Re-build | Deletes compiled files and launches Build task |
| 8 | Check | Launches the syntax checker task |
| 9 | Verify | Launches the verification task |
| 10 | Cross Reference | Constructs the proof obligation table |
| 11 | Stop | Stops current tasks |
| 12 | Clean | Cleans the Results window (20) |
| 13 | User Guide | Opens the user guide |
| 14 | Language Reference | Opens the language reference manual |
| 15 | Exit | Closes Perfect Developer |
| 16 | Add | Adds a pre-existing file to the project |
| 17 | Create | Creates a new file and adds it to the project |
| 18 | Remove | Removes the highlighted file from the project |
| 19 | Files Window | Shows the files in the current project |
| 20 | Results Window | Reports errors and progress of current task |

Table 2.1: Controlling Perfect Developer

subject of further investigation in Chapter 4.

### 2.4.3 The Compiler

The compiler translates Perfect source code into Java, C++ or Ada95 code. Executable code can be produced following compilation, by including a *"post-build"* command file that employs a third party compiler to further compile the compiled code into executable software. The compiled code relies upon runtime libraries, provided as part of Perfect Developer, which are undocumented and inaccessible to developers. Code is often *"mangled"*, whereby variable or method names are changed or method signatures altered, as a result of compilation.

### 2.4.4 The Verifier

The verifier is an automated symbolic executor and rewriting theorem prover that collects proof obligations of software and attempts to discharge them. It treats a specification as a monolithic entity and must prove all of the obligations of a source file in a single attempt. The verifier has two variables that control its execution, a memory limit and a boredom limit. The memory limit defines the maximum amount of memory a verification attempt is free to use. The boredom limit, limits the amount of time to be spent on a single verification attempt. These limits are used to prevent infinite application of the rewrite rules.

## 2.5 Conclusions

In this chapter, we have defined the concepts central to the thesis. Firstly, a selection of important software specification styles has been discussed. Secondly, some of the challenges faced by software implementations have been described. Thirdly, a collection of the most popular formal techniques for software verification have been defined. Lastly, the four components of Perfect Developer were presented to introduce the tool for comparison in the next chapter. The issues presented here will be kept at the forefront of our future investigations.

# Chapter 3

# Perfect Developer in comparison with Support Tools

In this chapter, a comparison of the Perfect Developer tool with similar automated support tools is presented. The chapter serves to illustrate some of Perfect Developers features and to fix its position relative to that of its peers. It also provides an interesting survey of automated tools. The tools are categorised by the verification technique they employ: *specification animation*; *extended static checking*; *refinement*; and *model verification*. Each tool is briefly described before being compared to Perfect Developer.

## 3.1 Specification Animation

Specification Animation allows users to pose questions about a specification to ensure that it is consistent and correct. A specification $S$ is accompanied with an assertion $P$ where $P$ in effect states that $S$ delivers a certain result for a certain input. The

animation of $P$ is comparable to a test run of $S$. Perfect Developer supports animation by encoding *properties* (See 4.5.3) alongside software specifications. Animators have been constructed for most specification languages, a selection of which is presented in this section.

### 3.1.1 Perfect Developer

Perfect Developer accepts specifications written in Perfect and verifies their correctness. Perfect is an object-oriented specification language that supports model-oriented and algebraic specifications. Perfect is relatively easy to learn, bearing a close resemblance to object-oriented languages like Java or C++. The developer may accompany specifications with algebraic properties, (assertions that define the correct behaviour of the software under construction, See 4.5.3) for specification animation. This property will be verified as part of the specification animation process. Verification proceeds automatically without any need for developer input and produces a complete proof of verified obligations. When verification is not possible, a proof output, suggesting reasons for the failure, is presented. Proof outputs may be in the form of html, text or TEX.

### 3.1.2 Z/EVES

The Z notation [99] is based on set theory and mathematical logic that supports model-oriented specification of software. It employs the language of *schemas* to describe mathematical objects and their properties. A schema describes the state of software, the potential state changes and the pre-conditions for a state change to occur. Specifications may be iteratively refined to other, more concrete Z specifications.

Z/EVES [70] assists in the construction, type checking and animation of Z specifications. Specifications can be animated through execution of the NEVER deduction agent [63]. NEVER is akin to a theorem prover, but requires more assistance from developers to perform animation. In this way it is thought to be quite similar to a proof checker [83].

While Z supports the refinement of specifications like Perfect, Z/EVES cannot be used to reason about these refinements in the way Perfect Developer does. NEVER is capable of advanced automated reasoning techniques, but rarely to the same scale or effect as Perfect Developer. The tool, and notation, requires high mathematical expertise from developers and is associated with a much steeper learning curve than Perfect Developer. While Z/EVES is a powerful tool for specification animation, it lacks the simplicity and features that make Perfect Developer a more attractive support tool.

### 3.1.3 CAFE

CafeOBJ [77], a successor to the OBJ, is an algebraic specification language that accommodates the object-oriented programming paradigm. This is achieved by combining the *order sorted logic* of OBJ with *rewriting logic*, that permits the definition of hidden sorts and concurrency. Order sorted logic allows types to be defined as `sorts`, where their behaviour is defined by a set of *sigma axioms*. Hidden sorts allow local attributes and state changing methods to be included in the definition of a type, thus describing a class. The specification mechanism is completed through a rewriting logic which allows equality and state transitions on hidden sorts to be defined. CAFE is the network based environment that supports systematic creation,

checking, verification, and maintenance of CafeOBJ specifications.

CAFE is still in the early stages of development and is currently driven by a command line user interface that is more challenging than Perfect Developers GUI. In previous experience with teaching CAFE and CafeOBJ, the tool was found to be fragile, collapsing during the animation of some simple specifications. Further to this, error reporting is of a poor quality and finding mistakes was pain-staking on even the smallest of source files. CafeOBJ is an implementable specification language, but implementations were extremely inefficient. Without the support for refinement that Perfect Developer provides, adequate implementations could not be obtained while ensuring correctness. CafeOBJ and CAFE offer an impressive mix of technologies but, as described, do not support software construction to the same extent as Perfect Developer

### 3.1.4 The VDM++ Toolbox

VDM++ [40] is a version of VDM (Vienna Development Method) [66] that combines model-oriented specification with the object-oriented programming paradigm. The specification language is supported by the VDM++ Toolbox [45]. The toolbox contains an interpreter that may be used to test specifications along with a coverage tool. The coverage tool measures the percentage of statements that are exercised by each operation during a trace [74].

The VDM++ Toolbox is a mathematically rich software development tool. It is closely related to Perfect Developer in offering software to be implemented in C++ by means of *dynamic link classes*. VDM++ is superior to Perfect in terms of richness

and the wealth of documentation currently available. However, the marriage of specification and object-orientation is awkward at times. The VDM++ Toolbox requires high mathematical expertise, making it less attractive to software developers than Perfect Developer.

## 3.2 Extended Static Checking

Extended static checking [31] is a technique that is designed to discover ordinary programming errors with software that are not usually known until runtime. The tools that support this technique are neither sound nor complete [37] but can be used during development to indicate common programming errors such as array index bound errors, null references and deadlocks in multi-threaded programs. It can be easily exploited by object-oriented programming languages that support design by contract lightweight specifications (See 2.1.4). Perfect Developer supports a form of extended static checking as part of its verification process. In this section we discuss four tools that employ extended static checking techniques, Perfect Developer, ESC/Java2, the KeY tool and Omnibus.

### 3.2.1 Perfect Developer

Perfect Developer supports extended static checking of contracts through what it terms "verified design by contract" [35]. Contracts are described using invariants, pre-conditions and post-assertions (these are post-conditions but Perfect uses different terminology, see 4.3.3). The verifier treats a class invariant as an implied

pre-condition to each method call and an implied post-assertion on each method declaration. The verifier collects proof obligations for each post-assertion that is declared in the software. For each call of a method, the pre-condition is collected as a proof obligation. These obligations must all be discharged in a single step. The entire process is automated and no interaction with the verifier is possible.

### 3.2.2 ESC/Java2

ESC/Java2 [80] is an extended static checker for Java programs. The tool is the successor to ESC/Java [31] that employs the Java Modelling Language (JML) [65] to encode specifications. Some components of JML are not supported by ESC/Java2, including frame conditions, deadlock assertions and race-conditions. The JML contracts that describe the behaviour of a method are presented to the Simplify theorem prover [37]. Simplify attempts to find failures in the specification rather than to verify them, but some specification failures can escape discovery.

ESC/Java2 is more attractive than Perfect Developer because of its target language is Java and therefore benefits from extensive existing support. The tool suffers, however, because it lacks soundness and completeness, two essential criteria for software verification. Perfect Developer appears to encompass a greater scale of software development, while ESC/Java2 appears to have a wider target audience.

### 3.2.3 The KeY Tool

The KeY Tool [8] is a software development tool that constructs JavaCard [13] implementations from UML designs [9] or JML specifications [65]. The tool is constructed

upon Borland Together software [2]. This may be seen as a disadvantage as a previous version of the tool failed to function correctly when Borland updated their software. A developer first constructs a UML class diagram with Together ControlCenter, then includes the OCL specification and finally constructs the JavaCard implementation. KeY utilizes a custom-built theorem prover, specialized to incorporate Harel's dynamic logic [47]. This theorem prover may automatically be used to verify the contracts of methods in the implementation.

This tool shares many similarities with Perfect Developer. Both are object-oriented supporting a development from UML to Java implementations, though Perfect Developer supports C++ and Ada95 implementations as well. Both employ a custom-built theorem prover with automatic verification capabilities, though the KeY theorem prover is technically documented [76] and interactive. The substantial differences are the support for refinement that is included with Perfect Developer and the scale of the Perfect specification language.

### 3.2.4 Omnibus

Omnibus [106] is a development environment and language that is currently under construction. The language incorporates runtime assertion checking, verified design by contract and extended static checking. The language is Java-like and aims to incorporate much of the Java GUI library. Currently the tool supports runtime assertion checking only, while ongoing work is developing extending static checking with the Simplify theorem prover and full verification with the PVS theorem prover. The tool began development in 2002 as part of an undergraduate project at the University of Stirling.

At the moment of writing, there is insufficient information about Omnibus to compare it fairly with Perfect Developer. The Omnibus language is Java like and has been used in the production of graphical software. However, this language has yet to be formalised and so offers limited security. Omnibus could be ranked in the same category as Perfect Developer when completed, but as yet too little is known about its details.

## 3.3 Refinement

Many developers question the value of a correct specification when the specification does not assist software development further. Refinement provides a correctness preserving link between a specification and its implementation. Tools that support this link, and the associated verification, provide the opportunity for producing correct, consistent and clear specifications that offer value in the short- and long-term. In this section, tools that include refinement as a key software development technique are presented.

### 3.3.1 Perfect Developer

Perfect permits a single refinement step from specification to implementation either by algorithm or data refinement. When a data refinement occurs, the concrete data is declared as the *internal variables* (See 4.7.1) and all methods of the class must be refined to reference only those internal variables. A *retrieve function* is used to define how to construct the abstract specification variables from the internal variables. Verification of a refinement occurs in whole alongside extended static checking and

specification animation. The implementation must be proved to meet its specification in its entirety; no segmentation of this proof obligation is possible.

### 3.3.2 The B Method

The B method is possibly the most widely recognised tool for producing large software that is formally verified. The method is composed of the B-method [7], a method which supports iterative refinement of specifications, and a B-tool, which may be Atelier-B [3], B4Free [30] or the B-Toolkit [10]. Software specifications are written in the *Abstract Machine Notation* (AMN) [7] which has its mathematical foundations in the theory of Generalised Substitutions. Software implementations are produced in standard C. The B method has been used by industry and academia extensively on projects across the world, most notably by Siemens in developing the Meteor driver less metro system [94] in Paris.

The B method with the B4Free tool is a powerful tool for software development through refinement. Unlike Perfect Developer, refinement is supported iteratively and through *retrieve relations* providing increased opportunity for refinement. However, this freedom comes at a high cost in expertise requirements. The B method has been repeatedly criticised for its complex mathematical nature [97]. The learning curve associated with B is much steeper than that with Perfect. Verification requires a lot more effort to get right as developers must have a deep understanding of the tool.

### 3.3.3 ProofPower

ProofPower enables the development of Z specifications and their refinement to Ada implementations. It is composed of a development environment, a compliance tool,

and the HOL theorem prover [43]. The development environment controls the construction and type checking of Z specifications. The compliance tool is used to specify and verify the Ada implementation. The HOL theorem prover performs verification in ML [104].

ProofPower is harder to use than Perfect Developer as it relies upon a combination of languages and technologies. Developers are provided with a more powerful verifier, but this weakens the tools appeal as requires a high level of expertise. While ProofPower is not nearly as difficult to use as the B method, it does offer a much greater challenge than Perfect Developer.

### 3.3.4 Specware

Specware is a design tool for the construction of provably correct software using iterative stepwise refinement [58]. Specifications are written in Metaslang [69], while implementations are produced in Lisp [78]. Metaslang is an algebraic specification language whereas Lisp is a powerful list processing language. Specifications are constructed from the bottom up, by *colimiting* or "glueing" smaller specifications together. Refinements, carried out in Metaslang, concretize the specification, defining the mechanism by which the specification may be satisfied. Once refinement is carried out, a Lisp implementation is generated and may be manually tested with Specware.

Specware provides good support for specification and implementation although it does come at a high cost. The Metaslang specification language is difficult to learn. Reliance upon LISP as the implementation language further reduces the appeal of the tool. The underlying verification technology is advanced, but lacks sufficient documentation to be thoroughly analysed. In comparison with Perfect Developer,

Specware is less attractive as a software development tool for general use in industry.

## 3.4 Model Verification

Model Verification is a completely automated reasoning technology that can prove software never enters dangerous or unreliable states. A model of software and a safety property is presented to a tool of this category which can ensure the property is valid in the model. Model Verification may take the form of *Model Checking* or *Model Finding*. Model Checkers construct a tree of possible paths of execution and test the safety property on each, ensuring all possible paths are explored. Model Finders attempt to refute the safety property by constructing instances of the model that break it, then deciding if these properties are valid within the software model. Each technique can be performed without any assistance from the developer. Perfect Developer does not support any form of model checking and therefore will not be directly compared with the following three tools.

### 3.4.1 U2B

U2B [96] integrates the UML and the B Method to construct a tool that compensates for the weaknesses of both. The tool assists developers in constructing B models of software under development by using the graphical UML diagrams to initiate the construction. Initial specifications written in UML are augmented with AMN specifications. The tool translates these specifications to pure B so that the B-Tool may verify their correctness. The tool simplifies the development of AMN specifications, a complex language that has met with many complaints from industry due to its high

mathematical expertise requirements [97].

The tool is still in the early stages of development at the University of Southhampton with many interesting problems still to be tackled. Object Oriented features like *Inheritance and Polymorphism* are yet to be included in the tool. However, the tool meets many of the needs of industry [97], e.g. providing formal support through an interface that doesn't require extensive mathematical expertise. Its main goal was to assist representation by simplifying the exploration and development of B specifications. While this goal is achieved, its support of software development is restricted to this.

### 3.4.2 SLAM

The SLAM [14] tool verifies the implementation of C device drivers for the Windows Operating System. The tool does not require any specification, but rather determines if an implementation will make an invalid call to the Windows kernel that may result in a system failure. It provides developers of device drivers with a *"black box"* that can provide a 100% guarantee on software correctness, and can pinpoint the location of mistakes precisely. The tool relies on the complete and correct specification of the Windows kernel to be available. While the tool has met with much interest at Microsoft, it will always struggle with the requirement that the operating system is completely and correctly specified [15].

The major weakness of SLAM is that it is only applicable to a small domain of software development problems. However, it illustrates the potential for support tools to provide complete, reliable and useful assistance with no additional development cost. The tool is completely practical and its application in all driver development

projects is strongly recommended. The tool provides no form of specification support by virtue of removing the need for the specification of components. The tool excels in its support of verification but its problem domain is too restricted.

### 3.4.3 Alloy Analyzer

The Alloy Analyzer [56] is a model finder which attempts to find a valid model that refutes a stated safety property. Models are developed in the *Alloy* specification language. Alloy is comparable to both Z and OCL, though less expressive than the former and better defined than the latter. It is based on first order logic and can model quantifiers, higher arity relations, polymorphism, subtyping, and signatures [56].

The Alloy Analyzer supports partial models to be analyzed making it particularly useful during development. It is fully automated therefore simplifying the associated learning curve. The Analyzer provides better verification of complex reactive software than the other tools explored. The Alloy language is very simple to learn and has been taught around the world in at least 15 Universities [75]. The Alloy Analyzer can be considered as one of the strongest tools which support model verification. However, the tool provides no support for implementation of software as there is no link between the Alloy specification and an implementation language.

## 3.5 Conclusions

In this chapter, Perfect Developer has been presented and its relative position in the domain of support tools outlined. In presenting this discussion, the current cutting

| *Tool* | *Specification* | *Implementation* | *Verification* |
|---|---|---|---|
| **Perfect Developer** | Perfect | Perfect | Auto Theorem Proving |
| **Z/EVES** | Z | N/A | Auto Theorem Proving |
| **CAFE** | CafeOBJ | CafeOBJ | Auto Symbolic Execution |
| **VDM++ Tools** | VDM++ | C++ | Man Theorem Proving |
| **ESC/JAVA2** | JML | Java | Auto Symbolic Execution |
| **KeY Tool** | UML+OCL | JavaCard | Auto Theorem Proving |
| **Omnibus** | Omnibus | Omnibus | Auto Symbolic Execution |
| **B** | AMN+GS | C | Man Theorem Proving |
| **ProofPower** | Z | Ada | Man Theorem Proving |
| **Specware** | Metaslang | Lisp | Man Theorem Proving |
| **U2B** | UML+AMN | N/A | Auto Model Checking |
| **SLAM** | N/A | C | Auto Model Checking |
| **Alloy Analyzer** | Alloy | N/A | Auto Model Finding |

Table 3.1: Summary of Support Mechanisms

edge in automated tools for formal software development has been surveyed. These investigations have covered a broad range of verification technologies that can be can be summarised in Table 3.1 while the quality of their support is summarised in Table 3.2.

Several interesting tools have emerged from this research. Both ESC/Java2 and the KeY tool appear to offer developers excellent support, but fall short in providing refinement. The B-method still appears to provide the best form of support for refinement, but suffers severely because of the high mathematical expertise it demands.

Perfect Developer offers a well-rounded tool that is easily accessible to developers while still providing refinement. The Perfect language is one of the easiest to learn, providing a rich specification language for development. Full automatic verification is not offered by any of the other tools in the category, though the extent to which verification is possible in Perfect Developer has to be explored. The tool is one of

| | *Specification* | *Implementation* | *Verification* |
|---|---|---|---|
| **Perfect Developer** | Excellent | Excellent | Good |
| **Z/EVES** | Excellent | N/A | Excellent |
| **CAFE** | Poor | Poor | Excellent |
| **VDM++ Tools** | Excellent | Poor | Average |
| **ESC/JAVA2** | Excellent | Poor | Good |
| **KeY Tool** | Excellent | Good | Excellent |
| **Omnibus** | Average | Average | N/A |
| **B** | Excellent | Good | Good |
| **ProofPower** | Good | Average | Good |
| **Specware** | Average | Average | Good |
| **U2B** | Poor | N/A | Good |
| **SLAM** | N/A | N/A | Excellent |
| **Alloy Analyzer** | Average | N/A | Excellent |

Table 3.2: Summary of Support Quality

the most advanced in its field offering, to a great extent, the elements required for a powerful software development tool.

# Chapter 4

# The Perfect Language

In this chapter, the Perfect specification/implementation language is presented. The chapter acts as a brief tutorial in Perfect rather than as a reference manual. Perfect is illustrated with the help of a worked example that highlights its syntax. The novel features of the language are described, for example the treatment of inheritance and polymorphism. For more information the reader is directed to the online reference manual [5]. A more extensive description of the language can be found in [26] and [27].

The example that supports this chapter is that of a harbour management system (See Figure 4.1). The software is for illustrative purposes only and will, therefore, be very simple. A harbour consists of multiple ports, at which ships may dock, and a queue, at which incoming ships await docking. A queued ship must be docked whenever there exists a free port. Ships are free to leave their port at any point in time. The software for this example is described by the UML class diagram in Figure 4.2 and defined in Appendix A.

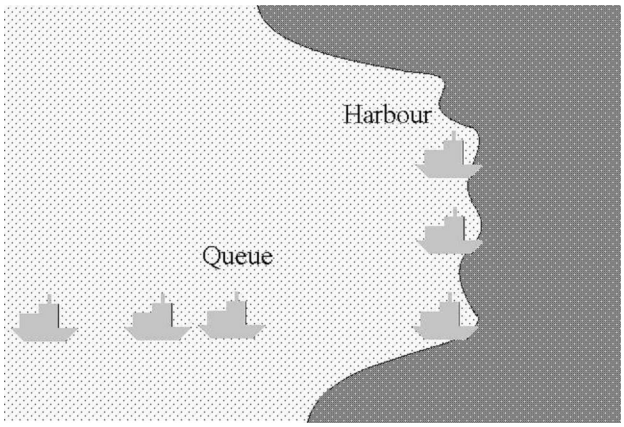


Figure 4.1: Harbour Example - Three Ships docked, Three Ships Queuing

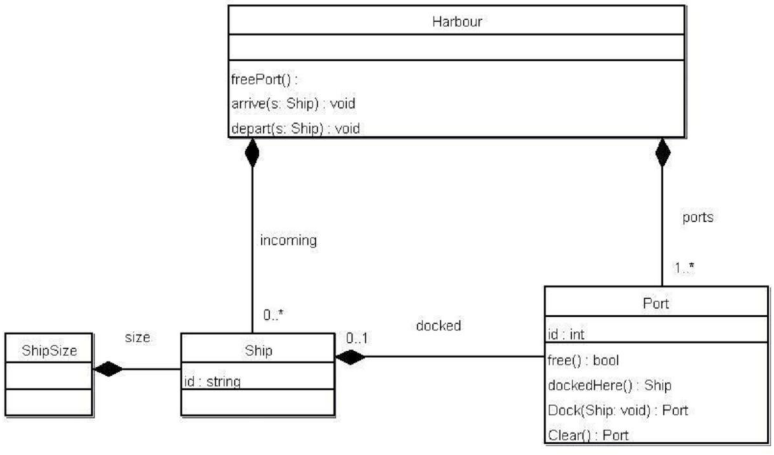


Figure 4.2: UML class diagram of the Harbour example

## 4.1 Typing

Perfect offers four mechanisms by which types may be introduced: *classes*; *enumeration types*; *constrained types*; and *generic classes*. All four are introduced by the keyword `class`. A class defines the attributes and behaviour of a set of objects, as previously described (See 2.2.2). An enumeration type defines a set of ordered values. A constrained type defines a set of allowed objects, constraining a previously defined type by some predicate. A generic class is a class that accepts types as parameters. In this section, each typing mechanism is defined in detail.

### 4.1.1 Class

Perfect supports the definition of classes in much the same way as other object-oriented programming languages. Within Perfect, however, visibility of class features is defined by defining features within *sections*, (i.e. a body of code). There are, at most, four sections in a Perfect class: `abstract`, `internal`, `confined` and `interface` where features may be defined. The data specification and the private methods of the class are defined within the `abstract` section. The data implementation and the retrieve function are defined within the `internal` section. Those methods accessible to the class and its descendants are defined in the `confined` section. Publicly accessible methods are defined in the `interface` section.

For example, the structure of the `Harbour` class will have the form:

```
class Harbour ^=
 abstract
    // Specification Attributes appear here
 internal
    // Implementation Attributes appear here
```

```
 confined
    // Private Methods appear here
 interface
    // Public Methods appear here
end;
```

### 4.1.2 Enumeration Type

An enumeration type defines a set of objects that are ordered by the elements location within the declaration. Enumeration types do not permit the definition of new methods, but support the generic comparison operator, `~~`. Application of the compare operation results in the relative `rank` of the object being evaluated. The enumeration type also has two class methods, (referred to as `non-member` methods in Perfect), that evaluate to the `highest` and `lowest` elements of the class.

To illustrate, consider that each ship in the Harbour example has an associated size. The set of ship sizes may be defined in an enumeration type as:

```
class ShipSize ^=
 enum
  Dinghy,
  Yacht,
  Barge,
  Ferry
end;
```

### 4.1.3 Constrained Type

A type may be restricted to a subset of possible instantiations through declaration of a constrained type. The constrained type defines a predicate such that only those instantiations for which the predicate holds are valid members of the constrained type. The constrained type contains all the features of the original type, but new

methods may not be declared. Polymorphism of the constrained type as the original type is permitted.

In the example, it may be of use to describe those ships whose size is strictly smaller than a Ferry. Using a Constrained type, the class is defined as:

```
class SmallShips ^= those s:Ship :- s.size < Ferry@ShipSize;
```

It is assumed the class `Ship` has a method called `size` that returns the size of a ship object.

### 4.1.4 Generic Classes

A generic class may be defined that allows the features of a class to be defined independently to the features type. The mechanism that permits this is referred to as parametric polymorphism [98], whereby type independence is resolved by including a type as a parameter to any instantiation of the class. The language of generics included in Perfect provides the developer with several options for defining restrictions on the type parameter. These include definitions of the equality and comparison operators on the independent type; inclusion of a specific method on the independent type; and inclusion of independent type within an inheritance hierarchy. Each of these restrictions helps to define the class, thus preventing abuse of the generic class.

In the Harbour example, it is required that the ships arriving at the Harbour are queued. Rather than developing a specific Queue class for ships to manage incoming ships, a generic `Queue of X` class could be developed and then re-used whenever a queue is required. The class would have the form:

```
class Queue of X ^=
abstract
```

```
 var queue:seq of X;
end;
```

This class would be instantiated in the Harbour system, by providing the type parameter `Ship` by:

```
 incoming: Queue of Ship != Queue of Ship{};
```

## 4.2 Class Attributes

Two levels of data attributes may exist within a class, one for specification and one for implementation. Both levels use the same syntax for declaring the attributes but are defined in different sections, as previously mentioned (See 4.1.1). Data attributes are introduced by the keyword `var` where an attribute identifier is defined followed by its type. To illustrate this, the `Ship` class is developed to include its data attributes.

```
class Ship ^=
 abstract
  var myName:string;  // Ship name
  var size:ShipSize;  // size
  var max:nat;        // max speed of the ship
```

As shorthand, variables may be defined within a single expression by replacing the intermediary semi-colons with commas. To illustrate this, the `Port` class is defined as:

```
class Port ^=
 abstract        // Specification of model
  var id:nat,               // the port id
      docked:Ship||void,    // a docked Ship or nothing
      used:nat;             // amount of times ships docked here
```

Perfect also provides a mechanism for constraining the values of variables within a class, through a design by contract class invariant (See 2.1.4). The `Harbour` class will require at least one `Port` object in its collection. This is captured by the data attribute specification:

```
class Harbour ^=
 abstract
  var ports:set of Port,        // all ports available
      incoming:Queue of Ship;  // incoming queue of ships

  invariant #ports >0;          // a harbour has ports
```

This example illustrates how a class may be populated with attributes in Perfect. These examples are all with respect to the `abstract` attributes of a class. These attributes are used to implement the software if data refinement has not been applied. When data refinement has been applied, the implementation attributes, located in the `internal` section, are defined in a similar manner. A class must have `abstract` attributes before `internal` attributes may be defined.

## 4.3 Class Methods

The methods of a class define the behaviour and message passing mechanism of the instances of the class. In Perfect there are three varieties of methods: *constructors*; *functions*; and *schemas*. Constructors define the creation of a new instance of a class. Functions are side-effect free methods that return a result without changing the global state. Schemas are state changing methods that return no result but may change the state of the current objects attributes or the state of parameters passed to the schema. Class methods, that do not require the class to be instantiated for

definition, may be declared as `non-member` functions.

### 4.3.1 Constructor

The constructor creates instances of the class by instantiating all of the attributes of the class and ensuring that all invariants hold. The constructor has a unique syntax in Perfect, introduced by the keyword `build`. Parameters are passed to the constructor inside curly braces { }.

A constructor that accepts values for all parameters will be written and placed in the `confined` section of the Port class to be accessible to the class and its descendants.

```
build{i:nat,d:Ship||void,u:nat}
 post id!=i,
      docked!=d,
      used!=u;
```

This constructor states, as its post-condition, that the value of `id` will become equal to the value of `i`, the value of `docked` will become equal to the value of `d` and the value of `used` will become equal to `u` when the method terminates. The three statements are equivalent to assignment statements. As the statements are separated by commas, the order of execution is not important, they may occur in any order or even in parallel.

A public constructor can be developed that takes advantage of this private constructor.

```
interface
// Global constructor
// All ports are created vacant and unused
 build{i:nat}
   ^= Port{i,null,0};
```

This constructor is equivalent to calling the previous constructor, with these standard parameters. Perfect offers a shorthand for constructors that allow the instantiation of attributes directly from the constructor parameters. To achieve this, the attribute identifiers are listed as the constructor's parameters preceded by a `!` symbol. This is illustrated by the constructor to the `Ship` class:

```
interface
 build{!myName:string,!size:ShipSize,!max:nat};
```

### 4.3.2 Function

A side-effect free method in Perfect is declared as a `function`. Functions must return a result based on an evaluation of the state of the object and the parameters passed to the function. It is often required to construct methods that return the value of an attribute without performing any calculation. These "*getter*" functions have the form:

```
function maxSpeed:nat
 ^= max;
```

This cumbersome syntax can be shortened in Perfect by stating the attribute name directly after the keyword `function`, for example:

```
function max;
```

Parameters to functions are listed within brackets, (), that follow the function name. Each parameter is required to have a name and a type, similar to parameters passed to the constructor. For example, a function in the harbour class that determines if a particular ship is docked at the harbour is written as:

```
function isDocked(s:Ship):bool
  ^= (let shipsDocked^= for b::ports yield b.dockedHere;
      s in shipsDocked
     );
```

The function creates a temporary set of all the ships docked in the harbour, using a `let` statement, and then tests if the parameter object is within the set.

### 4.3.3 Schema

Schemas define those methods that change the state of the software either by changing the local state of an object or by changing the state of the parameters of the schema. The keyword `schema` introduces these methods. The schema name is preceded by the `!` symbol when a local state change occurs. The `!` symbol also precedes the name of those parameters that may change during execution of the schema.

For example, it is expected that the Harbour will change its state whenever a ship arrives. The schema that defines the arrival of ships is declared as:

```
schema !arrive(s:Ship)
  post ([haveVacantPorts]:                // if vacant
              !land(s,vacantPort),          // land
        []:                               // else
           incoming!=incoming.append(s)     // add to queue
       );
```

It should be noted, that this schema contains a *conditional term*. The conditional term "if condition C1 then term t1 otherwise t2" is written as `[C1]:t1, []:t2`, where C1 is a boolean expression, and t1 and t2 are terms. Like constructors, the post-condition of a schema is declared as part of `post` expression. In this case, the schema declares that if the harbour contains any vacant ports, the ship is landed at one, otherwise the ship joins the incoming queue.

## 4.4 Assertions

An assertion is a boolean expression that should always evaluate to true. The expression, in Perfect, is introduced by the keyword `assert` and may be embedded within a code fragment at the specification or the implementation level. Assertions in Perfect are supported by a rich language to enable most statements to be clearly defined without significant effort. The language contains standard quantifiers and a selection of useful higher order functions that are presented below.

Universal and existential quantification is supported by Perfect across instances of the standard collection types (e.g. `set`, `seq`, `map`, or `bag`) or any other type, (with the exception of reference types, i.e. values declared on the `heap`). They are introduced by the keywords `forall` and `exists` respectively.

To illustrate, a declaration that all ships have a unique id is written as:

```
assert forall x,y:Ship :- x~=y ==> x.id ~= y.id;
```

Perfect also contains three of the common *higher order functions* from the functional programming paradigm. Higher order functions are functions that accept other functions as parameters. The included higher order functions are `fold`, `map` and `filter` which in Perfect are termed `over`, `for` and `those` respectively. A full description of their form can be found in the language reference manual [5].

## 4.5 Specification

The collection of specification techniques in Perfect that do not produce code are presented in this section. Lightweight specification through Design by Contract is supported in Perfect. Functions may be declared as *satisfy statements* to capture

the result of the function in a potentially non-deterministic predicate. Mathematical specification is supported by *algebraic properties* that state the requirements of the software abstractly.

### 4.5.1 Design by Contract

Perfect supports design by contract specifications with pre-conditions, post-assertions (referred to as post-conditions in traditional design by contract texts [72]) and invariants. Pre-conditions are introduced by the keyword `pre` and post-assertions are defined as regular assertions that are placed at the end of a method declaration. Both expressions may involve the values of the class attributes (at the specification level only) or their methods parameters. The post-assertion may reference the value of any of these variables at both the time of calling the method and at the methods termination. To distinguish between the variables, a priming of the variable name occurs to denote the final (or current) value of the variable.

To illustrate, consider the Harbour class and a method that defines the event of a ship leaving the harbour. The method, defined as a schema, assumes that that ship must already be docked. Once the ship departs, the method should guarantee two things, that it is no longer docked there and that it is not arriving at the harbour. The contract obligation is defined as:

```
schema !depart(s:Ship)
  pre isDocked(s)
   // An implementation will go here
  assert ~isDocked(s) & s ~in incoming;
```

Invariants of a class are always presented with the `invariant` keyword. The invariant is an assertion that may reference any attributes or functions that are declared

prior to the declaration of the invariant. Invariants may be broken by code fragments that are declared preceding the invariant within the source code. To illustrate a simple invariant, consider a restriction on our class that prevents ships from being both queued and docked. Such an invariant is defined as:

```
invariant ~(exists s::Ship :- isDocked(s) & s in incoming);
```

The invariant of the class may be arbitrarily complex, relying upon several helper functions that define the safety properties of the system. For example, a harbour should never allow a ship to queue while there is a port available. To include this as part of an invariant, the following definitions are required:

```
function vacantPorts:set of Port    // the ports available
   ^= those p::ports :- p.vacant;

function haveVacantPorts:bool      // are there vacant ports?
   ^= #vacantPorts >0;

function hasQueued:bool          // are there ships queued?
   ^= ~incoming.empty;

// if we have vacant ports, we don't have ships queuing
invariant haveVacantPorts ==> ~hasQueued;
```

As illustrated the assertion language of Perfect is ideally suited to Design by Contract providing quantifiers and permitting helper functions.

### 4.5.2 Satisfy Statements

One of the most unique aspects of Perfect are satisfy statements. These allow functions to be described by a predicate that defines the result abstractly. The statement is defined by including the keyword `satisfy` and the predicate after the function

declaration. The predicate must contain the keyword `result`, which refers to the value the function returns. These functions may be non-deterministic.

In the example, consider the function `vacantPort`, that returns a vacant port in the harbour. At the time of specification, there is no need to specify how to find this port. This method can therefore be specified as:

```
function vacantPort:Port
   satisfy result.isVacant;
```

### 4.5.3 Algebraic Properties

Perfect also allows an algebraic specification to be encoded alongside the object-oriented specifications. Algebraic properties can be used to define the behaviour of the class abstractly. They define the state that results from applying a sequence of methods to objects of the class. Algebraic properties may be used to capture software requirements during the early phase of specification. The keyword `property` declares the introduction of an algebraic property, which may list a set of parameters to the property and a pre-condition. The property is defined in an `assert` statement that defines the behaviour.

To illustrate the application of algebraic properties, consider the harbour example. It should always be the case that when a ship arrives and there are vacant ports, the ship will be docked instantly. This cannot be stated easily with contracts, but a property may be declared such that:

```
property(s:Ship)
    pre hasVacantPorts
    assert (self after it!arrive(s)).isDocked(s);
```

The `self` keyword refers to the current object. The `it` keyword refers to the object

referenced before the keyword `after`, in this case the `self` object. The property applies the `arrive` schema to the `it` object (i.e a copy of the self object). This resulting object calls the method `isDocked`. Stating this sequence as a property allows the semantics of the class to be defined clearly and concisely.

## 4.6 Inheritance and Polymorphism

Like most object-oriented languages, Perfect supports inheritance and polymorphism to a large extent. Both concepts are closely related in the object-oriented paradigm and are often the source of much debate, confusion and error [32]. Perfect overcomes these areas of difficulty by encoding a fault proof system of inheritance and polymorphism that will be presented below.

### 4.6.1 Inheritance

All classes in Perfect implicitly inherit from the `anything` class which provides the `toString` method. To explicitly inherit from a class, a child class is declared and followed by the keyword `inherits` and the parent class name. All the features of the parent class are included in the child class but only those features declared in the confined and interface sections are accessible. New features may be added and the old methods may be overridden or even removed.

Perfect requires the `redefine` keyword is used whenever a method is overridden. The specification of the overridden method must guaranteed by the new method. The pre-condition of the parent method may only be weakened by the new method and the post-assertion may only be strengthened by the new method. A function may

be hidden, through descendant hiding, by declared it `absurd`. Absurd functions are no longer methods of the class, but may be re-introduced later in the inheritance hierarchy.

Inheritance may be illustrated by adding a `FastShip` class that inherits from the `Ship` class. Suppose that all ships have an associated maximum speed and that all fast ships have a maximum speed above a certain limit. This would be encoded as:

```
class FastShip ^= inherits Ship
  interface
   build{n:string,s:ShipSize,m:nat} inherits Ship{n,s,m};

   redefine function maxSpeed:nat
     ^= max
    assert result > 50;
end;
```

Assume there is another class called `BrokenShip` representing ships that are damaged and can no longer move. It would be pointless to ask what the maximum speed of a broken ship is, so this would be declared as an `absurd` method.

```
class BrokenShip ^= inherits Ship
  interface
   build{n:string,s:ShipSize,m:nat} inherits Ship{n,s,m};

   absurd function maxSpeed:nat;
end;
```

Here it is impossible to make a call to `maxSpeed` to be made on all `BrokenShip` objects.

### 4.6.2 Polymorphism

Commonly in object-oriented languages, polymorphism through inheritance is supported by default. This is not the case in Perfect. Objects of descendant classes cannot be used whenever an attribute has the type of a parent class. This prevents errors that result from descendant hiding or specification adaptation occurring. Polymorphism through inheritance is, however, permitted, when the type of an attribute is decorated with the `from` keyword. For example, when a variable is declared as `from Ship`, the variable may reference an instance of type `Ship` or any of its descendants.

Perfect supports another form of polymorphism, referred to as *united types*. A united type is a type that is constructed from the union of two or more disjoint types. The operator of union is two vertical bars `||` and is commonly used to allow objects have a `null` value. For example, to permit a variable hold either a `Ship` or a `void` object, it is declared as:

```
var holder: void||Ship;
```

## 4.7 Refinement

The most novel concept in the Perfect language is its treatment of refinement. As has been discussed in 3.4, refinement is a formal process that permits the translation of specifications to implementations. It is supported within the Perfect language as a single refinement step from specification to implementation within a class. In this step, the developer may: refine a method specification to an implementation; refine a data structure through addition or transformation; or refine the specification itself

adapting it to accept more possibilities or guaranteeing a stronger system. Each form of refinement has been written about in depth in [28].

### 4.7.1 Data Refinement

*Data Refinement* is perhaps the most intriguing element of Perfect. It provides the user with a mechanism to enrich a specification with implementation details that, while unnecessary for the specification, are highly valuable to efficiency of the software. There are two forms of this refinement: *attribute introduction* and *type transformation*. Attribute introduction augments a specification with additional attributes whose purpose is to store values that are repeatedly derived in the software. Type transformation completely alters the data specification, introducing a new data structure in place of the one specified in a class. Data refinement has an impact on all methods. This impact is realised through algorithm refinement.

To illustrate type transformation, consider the collection of `Port` objects in a `Harbour` object. The `ports` have been specified as a `set of Port` for simplicity, but as regular accesses will be made either to those ports that are vacant or those ports that are in use, the `ports` attribute can be refined into two sets, one representing `vacantPorts`, the other representing `usedPorts`. The class is defined as:

```
class Harbour ^=
abstract        // Specification of model
 var ports:set of Port;     // all ports available

internal            // Data Refinement

 var vacantPorts:set of Port,     // free ports
     usedPorts:set of Port;       // ports in use
 function ports                   // retrieve function
   ^= vacantPorts++usedPorts;
```

```
// No port is free and in use
invariant vacantPorts**usedPorts=set of Port{};

// open ports are free & closed ports are not
invariant forall p::vacantPorts :- p.free;
invariant forall p::usedPorts :- ~p.free;
```

A *retrieve function*, `ports` is declared to define how the abstract variable is constructed from the internal variables. The retrieve function must be in the direction of internal to abstract only. Usually, the refinement requires invariants be declared to provide a semantics of the internal variables. This is required to perform verification of the refinement. The methods of the `Harbour` class must be algorithm refined as a result of the described type transformation.

### 4.7.2 Algorithm Refinement

Algorithm Refinement, as implemented in Perfect, permits a code fragment to be replaced with its implementation. Refinements may be made on the methods of a class or on any statements within a method. The refinement is introduced by the keyword `via`. Any satisfy statements declared in the class must be refined before compilation may occur.

To illustrate algorithm refinement, consider two methods of the `Harbour` class: a satisfy statement that returns a vacant port; and an isDocked method:

```
function vacantPort:Port
  satisfy result.vacant;
  via value openPorts.min
  end;

// is s docked in our harbour?
```

```
function isDocked(s:Ship):bool
  ^= (let shipsDocked^= for b::ports yield b.dockedHere;
      s in shipsDocked
     )
  via let shipsDocked^= for b::closedPorts yield b.dockedHere;
      value s in shipsDocked;
  end;
```

In the first method, the original specification offers no algorithm for finding a vacant port. The refined algorithm for finding a vacant port simply selects the least used port that is vacant. In the second method, the original specification implicitly defined an algorithm which searched through all the ports in the harbour. In the refined algorithm, the search space is reduced by removing those ports that have no ships docked at them.

## 4.8 Conclusions

This chapter has presented the core theory of the Perfect language. The language is an object oriented programming language that includes a rich assertion language enabling several styles of specification. It contains many novel features that set it apart, such as generics, higher-order functions, and refinement. The verifier is a useful feature for ensuring that software is coded correctly bringing many common programming errors to the developer's attention. On initial inspection, the language provides the developer with everything that could be required for formal software development, but further investigations into how well it scales up must be undertaken.

# Chapter 5

# Case Studies with Perfect Developer

In this chapter, developments of two medium scale case studies with Perfect Developer are described: A Library Database and a Resource Manager. The case studies were designed to illustrate a variety of common stumbling blocks for formal tools, database and reactive software. In each case study, we evaluate the support provided by Perfect Developer with respect to *specification*, *implementation*, and *verification* of the software.

## 5.1 Case Study 1: Library Database

Kemmerer's library database problem has been an illustrative tool for specification languages and development methods since 1985 [60]. Wing categorized 12 approaches to the problem [107] of developing a university library database. The database has two kinds of users, borrowers and staff and has the following transactions available

on the database:

- Add/remove a copy of a book to/from the library (staff only)
- Return/Borrow a copy to/from the library (staff only)
- Get the titles of books by some set of authors/subjects
- Find out a list of books currently borrowed by any user (staff only)/by yourself (user only).
- Find out which user last borrowed a copy (staff member only)

The database also has the following requirements:

- All items must be either checked out or available
- No item may be both checked out and available

In the original specification, it was assumed by the requirements that the Library would have some staff and some borrowers. We extended the specification to include operations to add or remove users to the software system. In a real system there would be some restrictions on these operations, but for the sake of brevity these restrictions are ignored.

### 5.1.1 UML

The structure of the Library Database is represented by the UML class diagram in Figure 5.1. The top level of the software is the `LibraryDB` class. This class distributes queries to the required modules (i.e. stock queries to the `LibraryStock` module and user queries to the `UserBase` module). The `LibraryStock` module manages a

`LibraryCatalog` that contains all the book details and the current stock levels of the library. This is represented as a collection of `Borrowing` instances. The `UserBase` manages the `BorrowerBase` and `StaffBase` which keep records of borrowers and staff respectively.

The skeleton code found in Appendix B was automatically generated by Perfect Developers *UML import feature*. This feature imports an .xmi file of the class diagram (as presented in Figure 5.1) into a new project created by the Project Manager. The skeleton code includes `?` symbols where specification or implementation details are required. The importer also generates several spurious source code files that represent the `nat` and `string` classes. All variable types in the skeleton code were defined as `from X` to provide polymorphism. This typing is an example of unnecessary additions to the software which could complicate verification at a later stated. The automatic declaration of attributes as selector functions has a similar effect. Multiple modifications of the generated skeleton code were required before development could progress.

A number of issues with Perfect Developer arose as a result of this case study. The extensive editing of the skeleton code made using the importer redundant. The cost of reviewing and correcting the code could easily be avoided by manually constructing the specification from the beginning which would probably be quicker in the long run. The importer, while a good idea, does not work well with Perfect Developer unless further information can be provided in the UML design (e.g. explicit typing of the attributes and methods). Another major problem is the lack of support to export software to an .xmi file. If the software is changed during the development, the original UML model will also require manual alterations.

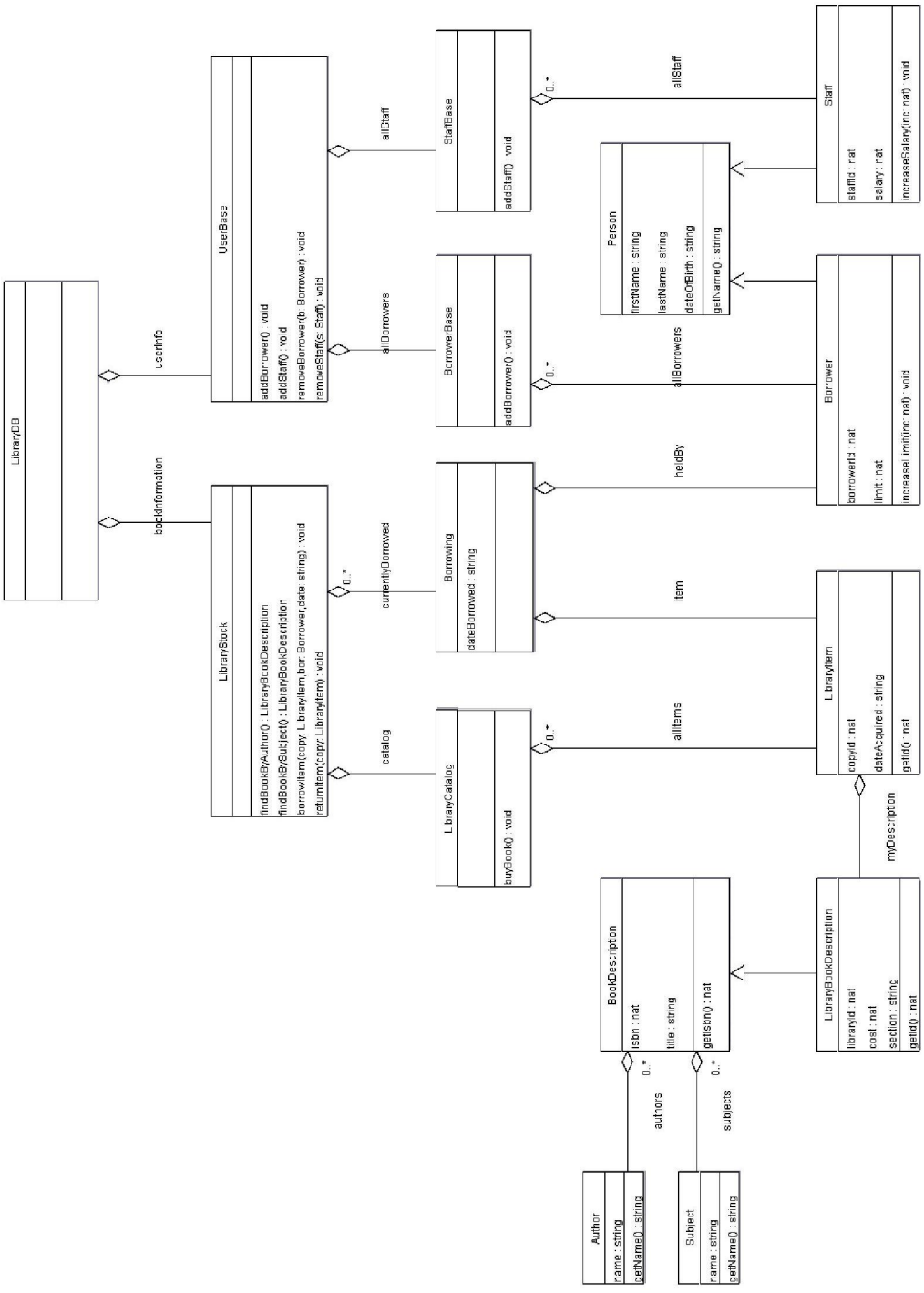


Figure 5.1: UML class diagram of the Library Database

### 5.1.2 Specification

The complete specification of the Library Database can be found in Appendix C, while here a selection of the most interesting specification are presented. This demonstrates the support for software specification that is provided by Perfect Developer.

The requirement that *"all items be either checked out or available"* is ensured by the structure that we imposed on the software. The set of items that are checked out are defined as a set of `Borrowing` objects. Therefore, the specification of `checkedOut` is:

```
function checkedOut:set of LibraryItem
    ^= for b::currentlyBorrowed yield b.getItem;
```

This has the benefit of defining the `available` items as those that the library has, less the items that are checked out:

```
function available:set of LibraryItem
    ^= catalog.allKnownItems -- checkedOut;
```

This simplifies the specification greatly ensuring the second requirement: that no item is both checked out and available. To ensure that only items known by the `LibraryDB` may be borrowed, an invariant is specified:

```
invariant forall a::currentlyBorrowed :- catalog.has(a.getItem);
```

The property that borrowing and returning an item has no net effect on the `LibraryDB` is ensured by the algebraic property:

```
property(i:Item,b:Borrower,d:Date)
  pre ~itemAvailable(i)
assert (self after it!borrowLibraryItem(i,b,d)
                then it!returnLibraryItem(i)) = self;
```

During the course of this case study, at least one way was found to specify the software requirements. Perfect is rich enough to capture the requirements at many levels ranging from the abstract properties to the specification of method contracts. Struggles with definition were rare, but occasionally appeared when nesting quantifiers. Overall, the support for representing the Library Database software could not be faulted.

### 5.1.3 File Handling

Permanent storage of any database is a necessary component to allow the software to terminate without losing all the information. Therefore, an implementation of the Library Database should incorporate this permanent storage issue. However, file handling in Perfect is inadequate providing limited features through an uncomfortable interface. File handling in Perfect is controlled through global access to an `Environment` object available through the GUI (See Appendix E.7). This object provides the ability to create, read and write to files directly.

The `Environment` class describes a `File` abstract data type, but this data type is unused in all of the file handling mechanisms. Instead, Perfect provides developers with file input/output via a `FileRef` object, (essentially a pointer to a block of memory). With this `FileRef`, the developer may read from a block of data interpreting the data as a raw byte or as a string. The data written to memory is treated as a sequence of bytes. Files have no structure in Perfect, and the language offers no means to provide it.

As a result of the weakness of the file handling mechanism in Perfect Developer,

including permanent storage of the database was abandoned in the software. Attempts were made to construct a customized `FileHandler` abstract data type (See 6.4.1). However, the underlying mechanisms provided very little power to ensure the correctness of the software under construction.

### 5.1.4 Refinement

Two data refinements of the library were carried out: one in the `LibraryStock` class and the other in the `LibraryCatalog` class. Both are printed as part of Appendix D. Only the latter is presented here for brevity. Within the `LibraryCatalog` class, the `catalog` variable is refined into a `HashedBucket` implementation.

```
internal
    var
        hashedLibraryItems:HashedBucket of LibraryItem;

    function allLibraryItems
        ^= hashedLibraryItems.ran;
```

This required the development of a generic `HashedBucket of X` class, the specification of which can be found in Appendix D.1. The development of the `HashedBucket` class required little effort because of the powerful specification language elements of Perfect. This produced an elegant and powerful generic class that could be integrated into the Library software with ease. However, it should be noted that the original specification of the `HashedBucket` had to be rewritten several times, though correct, to support verification of the refinement (See 5.1.7). Furthermore, the code representing the refinement is scattered throughout the source code file making it difficult to parse as specification or implementation alone. This mixture of specification and implementation is the only option offered by Perfect.

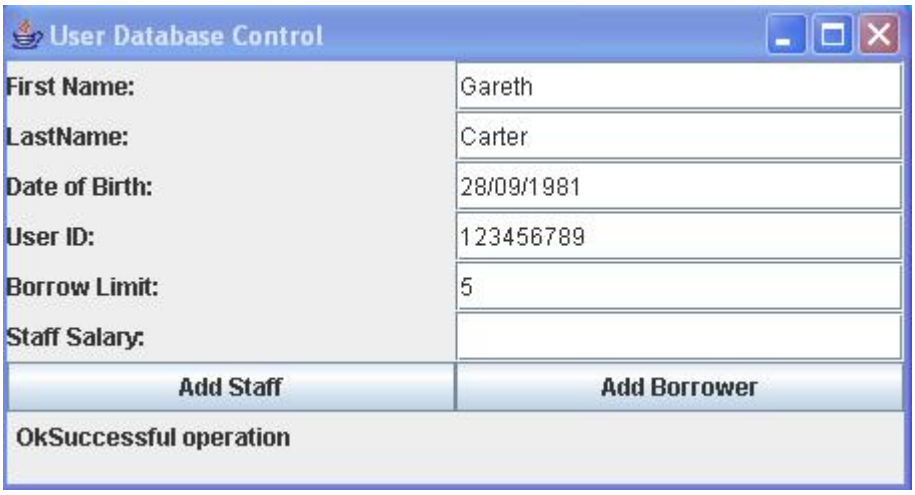


Figure 5.2: User Control GUI

### 5.1.5 Wrapper Class

Perfect Developer provides no mechanism to develop GUI's with Perfect. This results in the need to construct wrapper classes in the target language of the compilation. For this case study, the GUI is developed in Java and is presented in full in Appendix H.7. The wrapper class creates a GUI (See Figure 5.2, 5.3 and 5.4) that moderates the interactions between user and the Perfect implementation. The developer is required to read and understand the compiled Perfect code to properly develop a GUI.

Unfortunately, the compiled code is extremely difficult to parse as many of the functions and attributes are distorted from their original format. For example the `isNumber` function of the `LibraryDB` class was defined as:

```
function isNumber(val:string):bool
  ^= forall i::0..<#val :- val[i].isDigit & #val>0;
```

but has been compiled to a rather complicated piece of code which reads as:

```
public boolean isNumber (_eSeq val, char _t0val){
 boolean _vQuantifierResult_23_12;
```

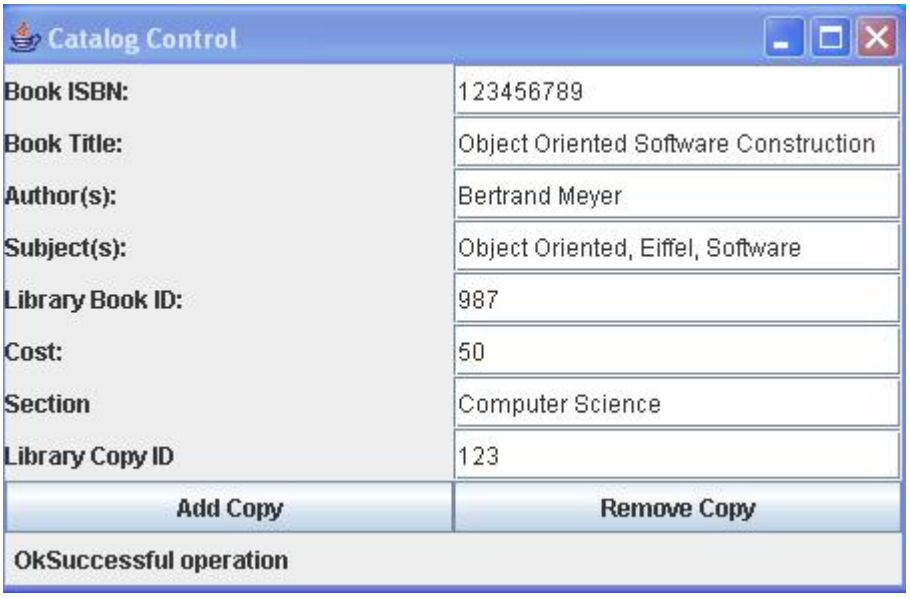


Figure 5.3: Catalog Control GUI

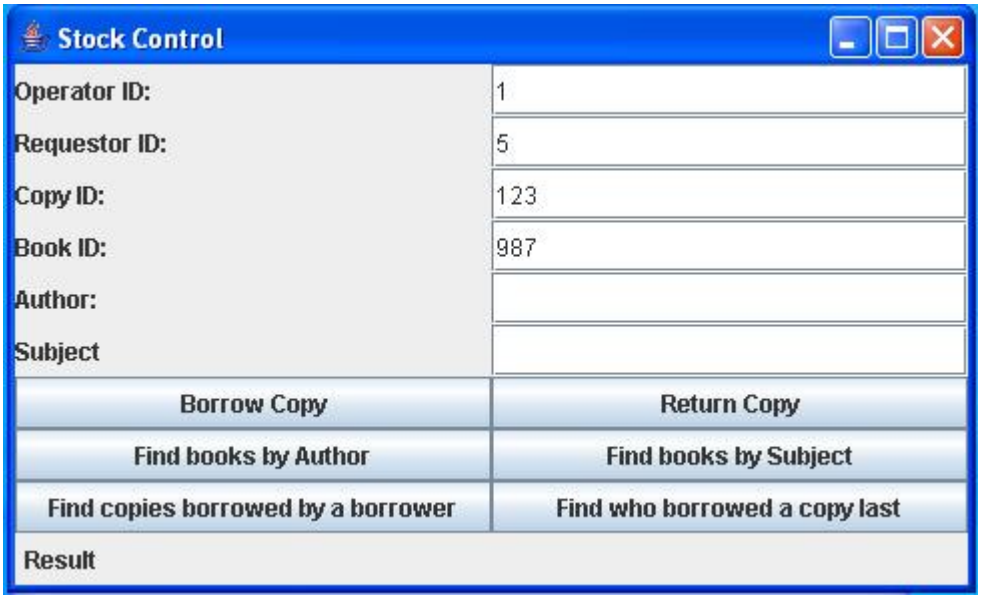


Figure 5.4: Library Database Control GUI

```
  {_vQuantifierResult_23_12 = true;
  int _vCaptureCount_i_23_26 = val._oHash ();
  int _vLoopCounter_23_19 = 0;
  for (;;) {
   if (((_vCaptureCount_i_23_26 <= _vLoopCounter_23_19)
                          || !_vQuantifierResult_23_12))
      ) break;
   _vQuantifierResult_23_12 = (
   _eSystem._lisDigit (
     ((_eWrapper_char) val._oIndex (_vLoopCounter_23_19)).value)
                                          && (0 < val._oHash ()));

    if ((!_vQuantifierResult_23_12)){

    }else{
       _vLoopCounter_23_19 =_eSystem._oSucc(_vLoopCounter_23_19);
    }
   }
  }
 return _vQuantifierResult_23_12;
}
```

The function signature has changed to include a new variable `_t0val` that is not referenced in the code, while the `string` type has become an `_eSeq` type. The `_eSeq` class is the Java class that provides the Perfect `seq` class functionality. Any `string` objects in Perfect will be considered as `_eSeq` objects in Java because the `string` class is defined as a `seq of nat`. The `char` variable, `_t0val`, denotes the type of the generic parameter of the sequence class. This requirement is not mentioned in the Perfect Developer documentation, but is needed in order to construct executable software.

It is noted that this mechanism could cause a software error if the developer interacts with the `_eSeq` object incorrectly. There exists no type checking of the elements in an `_eSeq` object. All elements are cast up to `_eAny` objects, Perfects

version of the `Object` type. This provides opportunities for `nat` objects to be inserted into `string` objects by accessing the Java code, (See 6.4.2). Although these difficulties hindered the software development, an implementation was eventually produced using Perfect Developer.

### 5.1.6 Animation

The Verifier struggled to animate the specification of the Library Management software. The main issues of contention are outlined here. A few errors, due to an incomplete specification, were discovered and corrected. The majority of verification errors were left unresolved by the verifier. Many of which arose out of trying to ensure the verification of the following invariant:

```
invariant forall item1,item2::allLibraryItems
                    :- item1~=item2 ==> item1.getId ~= item2.getId;
```

The verifier's main struggle arises when an item is borrowed or returned. When this occurs, the item must be deleted and a new object of the same id is inserted in its place. This unusual approach to changing state is necessary because Perfect Developer uses *value semantics*, whereby attributes contain unique objects rather than references. Ultimately, Perfect Developer verified 83% of the software, failing to prove 35 out of 196 proof obligations. It is interesting to note that all of the failed obligations could be proven manually.

### 5.1.7 Refinement Verification

The verification of the refined software was poor with only 63% of proof obligations being correctly discharged by Perfect Developer. This failure was due to a weakness

of the theorem prover which requires the specification of the internal data type be made in the same style as the specification of the abstract data type. For example, the following refinement could only be verified by altering the specification of the `add` schema in the `HashedBucket` class:

```
schema !addLibraryItem(copy:LibraryItem)
    pre ~hasItem(copy),
        uniqueItemId(copy.getId)
    post allLibraryItems!=allLibraryItems.append(copy)
    via
        allLibItems!add(copy)
    end;
```

The specification of `add` had to be modified to mirror the specification of `append` in the Perfect Developer Reference Manual [5]. Even though the specification of `add` was satisfactory to verify the `HashedBucket` class, the modification allowed this specification to be verified.

Other specifications could not be changed as easily. The complement to the above schema, the `removLibraryItem` schema, could not be verified even when an identical specification was provided. Through correspondence with Escher Technologies, it was discovered that the specification provided in the Reference Manual was incomplete as additional specification properties were not included.

Much work went into exploring Perfect Developers support of refinement. In particular, the theorem prover was found to be weak when refinement between two distinct abstract data types occurred. These refinement experiments are documented in [28]. In this paper, an array of experiments with Perfect Developer was undertaken with respect to refinement. The examples illustrate the traditional techniques of algorithm and data refinement, while also introducing the novel technique of *delta*

*refinement*, which allows specification change to occur through refinement.

### 5.1.8 Observations from Case Study 1

Perfect Developer supported the development of the Library Database well but there were limitations. Perfect provided good support for both specification and implementation although the construction of an executable piece of software was difficult. The need for wrapper classes makes development difficult as two development languages are required. The lack of adequate file handling mechanisms limits the use of Perfect in industrial software and the tools integration with UML is poor, requiring developers to iterate through many revisions.

Most disappointing was the support for reasoning in Perfect Developer. The verifier is claimed as the strongest component of Perfect Developer, yet it is the component most lacking in documentation and user-friendliness. The information provided when proofs fail is difficult to read and often relies on sub-proofs that contain variables not mentioned in the original goal. These problems were evident in both the specification and implementation verification, but were most problematic after refinement when no supporting evidence is given for the reason failures occur.

## 5.2 Case Study 2: Resource Manager

A Resource Manager is an operating system component that is in charge of allocation and de-allocation of resources to processes. Its job is essential in the operation of multi-threaded systems where many processes could be running simultaneously each making ongoing requests for resources to complete their jobs. Ideally, a resource

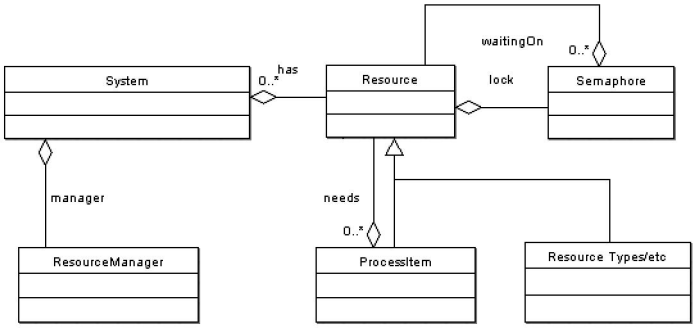


Figure 5.5: UML class diagram of the Resource Manager

manager should prevent deadlock or livelock, permitting all processes to complete in an efficient and timely manner.

The aim of the case study was to experiment with Perfect Developers verification technology. Equally important to the discussion is how the tool deals with reactive software problems.

### 5.2.1 UML

The high level structure of the Resource Manager is presented in Figure 5.5. The `System` class takes on the role of resource management. It is aware of all the resources and their state in the system spawning processes, accepting their requests for resources and granting the resources on the judgment of the `manger`. The `manager` is a resource manager that contains the algorithms that determine whether or not a resource should be granted to a process. Control of the resources is controlled by the use of a `Semaphore` object.

As with the Library Manager, the UML class diagram was imported to Perfect Developer and edited as required. The full specification can be found in Appendix F and the following discussions of the case study will highlight the elements of most

significance to this study.

### 5.2.2 Temporal Specification

Unfortunately, specification of the Resource Manager is almost impossible as Perfect contains no language for *temporal specification* [86]. It would be beneficial to represent that a process will *eventually* acquire all its resources or that all process *eventually* get executed, but neither is supported. Attempts were made to include this form of specifications by defining the notion of `progress`. Progress defines how close a process is to termination by measuring the number of resources that are required for execution and how soon it is believed these resources will be acquired. By defining `progress` for each `Process` object, a `System` specification that defines eventual execution of all processes could be encoded as:

```
 property(i:nat)
assert (self after it!step(i)).progress < self.progress;
```

### 5.2.3 Safety Properties

The essential safety properties that a resource manager should maintain are that deadlock and starvation never occur in the system. It was expected that these two properties could be specified by Perfect in the software. Deadlock was defined as:

```
 function hasDeadlock:bool
  ^=self after it!step.progress >= self.progress;
```

This was of no use in verification and was replaced by:

```
// if there are no resources, there is no deadlock otherwise
// deadlock occurs if the set of resources contains a cycle
```

```
function hasDeadlock:bool
  ^=([resources.empty]:
       false,
     []:
       exists r::resources :-
            hasCycle(set of from ResourceItem{},r,r.next.rep(1))
  );

function hasCycle(viewed:set of from ResourceItem,
                  at:from ResourceItem,
                  toBeViewed:bag of from ResourceItem):bool

  pre viewed <<= resources,
      at in resources,
      toBeViewed.ran <<= resources
  decrease #resources - #viewed
  ^= ([toBeViewed.empty]:
        false,
      [at in viewed]:
        true,
      []: (let newAt^=toBeViewed.min;
        hasCycle(viewed.append(at),newAt,
                   toBeViewed.remove(newAt)++newAt.next.rep(1))
      )
  );
```

The specification of starvation is impossible but could be approximated by including a `timeToStarvation` variable associated with each process. The System could then define an invariant, such that:

```
invariant forall p:ProcessItem :- p.timeToStarvation > 0;
```

This approach is highly application dependent as the `timeToStarvation` value would differ from system to system. Each new instance of the resource manager would require immense experimentation to discover good `timeToStarvation` values. Ensuring this invariant would require the Resource Manager to check all processes and

would be highly inefficient to guarantee. The specification of this problem appears to be beyond the capability of Perfect Developer to specify.

### 5.2.4 Parallel Execution

Perfect does not support multi-threaded applications although this case study is inherently multi-threaded. An approximation of multi-threading was therefore included by developing a `step` method that behaves like a system clock and performs some action determined by a number provided to the method. By assuming this number is randomized, a model of parallel execution can be brought into existence. The `step` schema is structured so that:

```
schema !step(choice:nat)
 pre ~systemEnd      // there is work to do.
 post (var p:ProcessItem,r:from ResourceItem;
   [choice=0]:
        If a process needs a resource,
        allow it to declare this need.
      ),
   [choice=1]:
        If a process is currently waiting on
        a resource available, allocate it.
      ),
   []:
       If a resource is currently held by
       some process, release it.
      )
);
```

If `choice = 0`, a random hungry process, called `p`, is selected from the set of executing processes that require resources. A random resource needed by this process, called `r`, is selected and removed from the set of resources. The `manager` attempts to acquire resource `r` for the process `p`. The objects are then returned to the system

and execution continues. If the system has no hungry processes, this step is ignored.

```
[choice=0]:   // declare want of resource
   ([hasHungry]:
       p!=pickHungry then
       resources!=resources.remove(p) then
       toBeExecuted!=toBeExecuted.remove(p) then
       r!=p.getNeed then
       resources!=resources.remove(r) then
       manager.acquire(p!,r!) then
       resources!=resources.append(p) then
       toBeExecuted!=toBeExecuted.append(p) then
       resources!=resources.append(r),
    []:
       pass    // System has allocated all it needs
               // we don't need to acquire any more
   ),
```

The other choices available in the `step` schema can be viewed in Appendix F.7.

### 5.2.5 Value Semantics

Perfect employs value semantics as part of the implementation language. While the choice simplifies verification of software, removing aliasing from occurring, it complicates software implementation. As illustrated, code expansion occurs when variables are required to change state. During execution of the `step` schema, the values of the resource and process objects may change. These objects are cloned from objects of the `resources` and `toBeExecuted` sets, rather than as references to the objects found there. For this reason, the objects must be removed from the sets, changed, and then re-inserted into the now cloned sets. This not only is a cumbersome policy of data overriding but an inefficient one, which requires searching the sets and repeated full cloning of objects.

### 5.2.6 Non-Termination

The Resource Manager doe not need to terminate while there are processes being created that have needs. This notion of potential infinite execution cannot be modelled in Perfect Developer because the tool requires *total correctness* of verification. Total correctness of a component demands the component guarantees its post-condition at termination and that it terminates at some point in the future. It is enforced in Perfect Developer by defining a variant in all recursive functions and loops.

### 5.2.7 Animation

The verification of the Resource Manager case study is extremely difficult to achieve. A complete verification was impossible. Influencing factors include the lack of elegant specifications and a weakness of the verifier when proving implications.

The specification language of Perfect Developer was not designed to specify temporal quantities, but an attempt was made to approximate them during this case study. These specifications are inelegant and cause the verifier great difficulty in attempts to manipulate them.

Perfect Developer does not allow verification by induction, a major disadvantage to this case study. The tool opts for a brute force strategy that attempts all possible combinations when trying to dispatch a proof obligation. This approach suffers with the Resource Manager case study, as it may permit a huge range of potential situations to arise, even on small datasets. The deadlock and starvation properties were included for verification with small datasets, but could not be verified by Perfect Developer. This case study showed a weakness in the verification support provided by Perfect Developer.

### 5.2.8 Observations from Case Study 2

The representation and implementation of a resource manager is beyond the capabilities of Perfect Developer. Perfect contains none of the elements of temporal specification or multi-threading making this task almost impossible. An approximation of the problem was developed, illustrating the power of Perfect as a specification language. However, it would be unfair to criticize Perfect Developer for not providing verification support for this case study because it was never intended to work for this software exemplar. After initial investigations, this case study was deemed unsuitable for deeper study. Perfect Developer provides no support for reactive systems and must be regarded as a tool for the development of single threaded software only.

## 5.3 Conclusions

In this chapter, two medium scale case studies with Perfect Developer were presented. These case studies have generated mixed results. Specification is well supported by the Perfect language. In each case study, there existed some way to specify what was required in the software. Unfortunately, both implementation and verification of software with Perfect Developer seems to suffer on larger scale software. File handling with Perfect is inadequate and multi-threading is not supported at all. Making operations efficient is considered wasted time as the extensive cloning, that result from value semantics, makes operational efficiency trivial. The verification suffers from poor documentation, resulting in wasted time correcting specifications that are correct but not specified in a style acceptable to the verifier.

# Chapter 6

# Illustrative Examples

In this chapter, several small scale examples with Perfect are presented. The chapter may be treated as an optional in that it does not describe or evaluate the results of examples in detail. However, most of the examples contribute to the analysis of Perfect Developer (in Chapter 7) as they highlight interesting experiences with the tool. These examples were driven by limitations encountered during the larger case studies described in Chapter 5. The examples cover the topics of *refinement*, *inheritance and polymorphism*, *generics* and *external interactions*.

## 6.1 Refinement

This section illustrates the techniques of algorithm and data refinement in Perfect Developer. In each case, an isolated problem is described and correctly refined.

### 6.1.1 Algorithm Refinement

In this example, a refinement of a `sort` specification to an insertion sort implementation is presented.

```
// Defines that two sequences are a permutation if, when ordering
// is removed both are equal
function isPermutation(a:seq of int,
                       b:seq of int):bool
   ^= a.ranb=b.ranb;

// Defines that a sequence is sorted if  every element is less
// than or equal to every subsequent element
function isSorted(a:seq of int):bool
 decrease #a
  ^=([a.empty]:true,
     []:(forall e::a.tail :- a.head<=e)
         & isSorted(a.tail)
     );

// Defines the result of sort as a permutation of the input that
// is sorted. Implemented as an insertion sort algorithm
function sort(ip:seq of int):seq of int
  satisfy isPermutation(result, ip),
          isSorted(result)
  via
   var op:seq of int!=seq of int{};
   loop
    change op
    keep 0<=#op'<=#ip,
      isPermutation(op',ip.take(#op')),
      isSorted(op')
    until #op'=#ip
    decrease #ip-#op';
     op!=addInPlace(op,ip[#op]);
   end;
   value op;
  end;
```

```
// Adds the element b in the sorted position of the sequence a
function addInPlace(a:seq of int,
                    b:int):seq of int
 pre isSorted(a)
 decrease #a
  ^=([a.empty]:
  // Insert item if an empty sequence
      a.prepend(b),
     [b > a.head]:
  // Recurse if not in correct position
      addInPlace(a.tail,b)
                     .prepend(a.head),
     []:
  // Insert in the correct position
      a.prepend(b)
    )
 assert isSorted(result),
    isPermutation(result,a.prepend(b));
```

Even though the algorithm is remarkably straightforward, the verifier still struggles to prove it correct. The resulting output is available in Appendix G. The failure is because the verifier does not acknowledge that two sequences which are permutations of each other remain permutations when the same element is added to both. This was defined as the axiom:

```
axiom assert forall a,b,c:seq of int, d:int :- isPermuration (a,b),
     isPermutation(c,a.prepend(d)) ==> isPermutation(c,b.prepend(d));
```

Adding this axiom to the specification doesn't result in a successful verification. The axiom, it appears, is not used by the verifier during the verification.

### 6.1.2 Data Refinement

In this example, the seemingly trivial refinement of the Perfect data type `set` is refined by the `seq` data type.

```
// A class that represents sets but only provides the add and remove
// schema. Implemented as a sequence
class SetSeq ^=
abstract
 var setofNums: set of int;

internal
 var seqOfNums:seq of int;
 function setofNums
    ^= seqOfNums.ran;

interface

// Equality definition
 operator =(arg);

 build{}
   post setofNums!=set of int{}
   via  seqOfNums!=seq of int{}
   end;

 schema !insert(a:int)
   post setofNums!=setofNums.append(a)
   via  seqOfNums!=seqOfNums.append(a)
   end;

 schema !remove(a:int)
   post setofNums!=setofNums.remove(a)
   via
    loop
     var i:nat!=0;
     change   seqOfNums
     keep i'<=#seqOfNums',
        a ~in seqOfNums'.take(i'),
        seqOfNums'.ran<<=seqOfNums.ran,
        seqOfNums.ran=seqOfNums'.ran |
        seqOfNums.ran =
               seqOfNums'.ran.append(a)
     until    i'=#seqOfNums'
     decrease #seqOfNums' - i';
```

```
      if
       [a=seqOfNums[i]]:
        seqOfNums!=seqOfNums.remove(i);
       []:
        i!=i+1;
      fi;
    end;
   end;
end;
```

The discovery of a correct verification of this refinement occurred through arduous experimentation with Perfect Developer. Multiple refinements were attempted, but only the one included here was successfully proved by the verifier.

### 6.1.3 Observations

One of the most admirable techniques adopted by Perfect Developer is refinement, yet the verifier often fails to prove correct refinements. The developer can never be sure if a failure to verify is due to a mistake or just verification limitation because of the verifier is poorly documented. To achieve a successful verification, the developer is forced to write implementations that stylistically resemble their specification. The language, it seems, is more expressive than the verifier will permit.

## 6.2 Inheritance and Polymorphism

This section illustrates Perfects treatment of Inheritance and Polymorphism. The first example documents the Perfect Dynamic Binding signature [19]. The second example illustrates how to overcome no-variance in Perfect by utilizing *functional contracts*. The third example demonstrates how the strict treatment of inheritance

in Perfect sacrifices some of the richness of the object-oriented paradigm.

### 6.2.1 Dynamic Binding

Using Beugnard's process [19], a dynamic binding signature for Perfect was identified. The full range of potential bindings is developed using the inheritance hierarchy of `Top`, `Middle` and `Bottom`. This is presented to two classes, `Up` and `Down`, that demonstrate the co-variant, contra-variant and no-variant bindings.

```
// An empty class representing the top-most element of an
// inheritance hierarchy. Used to test for contra-variance
class Top ^=
interface
build{};
end;

// An empty class representing the middle element in an
// inheritance hierarchy. Used to test for no-variance
class Middle ^=
inherits Top
interface
build{} inherits Top{};
end;

// An empty class representing the bottom-most element in an
// inheritance hierarchy. Used to test for co-variance
class Bottom ^=
inherits Middle
interface
build{} inherits Middle{};
end;

// A class that tests for co-, contra- and inv- variance
class Up ^=
interface
       build{};
```

```
       function cv(t: from Top) : string
pre t within from Top
^= "up";

function ctv(b:from Top):string
pre b within from Bottom
^= "up";

function inv(m:from Top):string
pre m within from Middle
^= "up";
end;

// A descendant class that tests for co-, contra- and inv- variance
// Used to test polymorphism in the test call.
class Down ^=
inherits Up
interface
build{} inherits Up{};

redefine function cv(m:from Top):string
pre m within from Middle
^= "down";

redefine function ctv(m:from Top):string
pre m within from Middle
^= "down";

redefine function inv(m:from Top):string
pre m within from Middle
^= "down";

end;
```

The results of the compilation, verification and runtime execution of the software is presented in Table 6.1. Compilation and verification errors are marked as “error” while “up” and “down” signify the printed output of the method execution.

| calls | u | d | ud |
|---|---|---|---|
| cv(t) | up | error | error |
| cv(m) | up | down | down |
| cv(b) | up | down | down |
| ctv(t) | error | error | error |
| ctv(m) | error | down | down |
| ctv(b) | up | down | down |
| inv(t) | error | error | error |
| inv(m) | up | down | down |
| inv(b) | up | down | down |

Table 6.1: Dynamic Binding Signature of Perfect

This signature is not equal to the ideal dynamic signature that Beugnard would predict, but is closer than other object-oriented languages.

### 6.2.2 Inheritance as Specialisation

This example illustrates the application of inheritance to perform class specialisation. Specialisation is commonly used as the primary justification of inheritance [102], but unfortunately its application is not type sound. The example shows the danger of constructing a Vegetarian who "is-a" Person, but cannot eat meat.

```
// A class representing all types of food, distinguished by name
class Food ^=
abstract
       var foodName:string;
interface
    function foodName;
    build{!foodName:string};
end;

// A descendant class that represents all types of vegetables.
// Vegetables are the only food that may be eaten by vegetarians.
class Vegetable ^=
```

```
inherits Food
abstract
interface
    build{vName:string} inherits Food{vName};
end;

// A class that represents all the People, distinguished by name
// People may eat anything, once it is food
class Person ^=
abstract
 var personName:string;
interface
build{!personName:string};

function personName;

function eat(f:from Food):string
  f within Food
^= personName ++ " has eaten "++ f.foodName;
end;

// A descendant class, in the "Vegetarian is-a Person" category
// Vegetarians, however, can only eat vegetables. This
// creates a problem for the type system
class Vegetarian ^=
inherits Person
abstract

interface
build{pName:string}inherits Person{pName};

absurd function eat(f:from Food):string
  false;

function eatDifferently(f:Vegetable):string
^= personName ++ " has eaten "++ f.foodName;
end;
```

Perfect does not allow overloading methods when the type which distinguishes between variables is a descendant type. This forces Vegetarians to use a different `eat` method, limiting re-use of classes.

### 6.2.3 Co-variance and Contra-variance

In this example, the Perfect treatment of variance through inheritance [29] is presented. This treatment is demonstrated by a Point-to-ColourPoint inheritance relation and the `equals` method. The class uses functional contracts to overcome the no-variance that was presented in the previous example.

```
// A class representing a point in 2d space
class Point ^=
abstract
    var x,y:int;
interface
build{a,b:int}
post x!=a, y!=b;

function getX:int
^= x;
function getY:int
^= y;

function likeCurrent(p:Point):bool
^= p within from Point;

function equals(p:from Point):bool
pre likeCurrent(p)
^= (p is Point).getX=getX &
   (p is Point).getY=getY;
end;
```

A class representing 2d points that also have a colour is developed. This illustrates the functional pre-condition `likeCurrent`. It also shows how binary methods may

be dangerous without this control mechanism.

```
class ColourPoint ^= inherits Point
abstract
    var
colour:int;
interface
build{a,b,c:int} inherits Point{a,b}
post colour!=c;

function getColour:int
^= colour;

redefine function likeCurrent(p:from Point):bool
^= p within from ColourPoint;

redefine function equals(p:from Point):bool
pre likeCurrent(p)
^= (p is ColourPoint).getX=getX &
   (p is ColourPoint).getY=getY &
   (p is ColourPoint).getColour=getColour;
end;
```

The functional contract `likeCurrent` can be used to provide method adaptation by pre-condition strengthening or similarly by post-condition weakening. This provides more freedom to developers to construct specifications in Perfect. The verifier behaves well in accepting these functional contracts.

### 6.2.4 Observations

Perfect Developer does not settle for a single unified theory of inheritance, but instead mixes and matches to provide different options. In some sense, the language is novariant in its requirements of very strict typing. The method specifications, however,

are co-variant, requiring that each overridden method implements its ancestral specification. This theory is further diluted by allowing descendant hiding and functional contracts that permit the specifications to contra-variant. The Perfect language attempts to overcome all the weaknesses that arise with inheritance and polymorphism but lose the elegance of theory in this attempt.

## 6.3 Generics

Perfect has a strong language of generics that enables abstract classes to be constructed much like template classes in C++ [16].

### 6.3.1 Non-empty Sequences

This example illustrates Perfects language of generics. It defines a class representing those sequences that are not empty, providing a mix of common and interesting features associated with the class. Ideally, the class could have been defined using a *constrained type* (see 4.1.3), but in that case new features could not be defined.

```
// This is a class with its own abstract data and operations
class NonEmptySequence of X require X has operator =(arg);
                                        total operator ~~(arg) end ^=
abstract
    var m:seq of X;
    invariant #m>0;

interface

// Construct using a non-empty sequence
    build{aseq:seq of X}
       pre ~aseq.empty
       post m!=aseq;
```

A method that reverses the non-empty sequence is developed:

```
// construct using a single element
    build{a:X}
       post m!=seq of X{a};

// reverses the non-empty sequence
    function reverse :  NonEmptySequence of X
    satisfy #result = #m, forall i::m.dom :- result[i] = m[#m-i-1]
    via
        var b : seq of X!=seq of X{};
        loop
            var n : nat !=0;
            change b
            keep n'=#b', n'<=#m,
            forall i :: 0..<n' :- b'[i] = m[#m-i-1]
            until n' = #m
            decrease #m - n';
            b!= b.append(m[#m-n-1]);
            n! = n + 1;
         end;
         value NonEmptySequence of X{b}
    end;
```

A function that determines if the current non-empty string is a palindrome (i.e. it may be written the same forwards as backwards) is developed. Attempts were made to specify the methods differently, but failed due to weaknesses on the part of the verifier.

```
    function isPalindrome : bool
         ^= forall i::m.dom :- m[i] = m[#m-i-1]
     via
        var b : bool != true;
        loop
            var n : (those x: nat :- x <= #m/2) != 0;

            // define a constraint directly on the type of n
```

```
            change b
            keep  0<= n' <= #m/2,
            b' = (forall i :: 0..<n' :- m[i] = m[#m-i-1])
            until (n' = #m/2) | ~b
            decrease #m - n';
            b! = (b & (m[n] = m[#m-n-1]));
            n! = n + 1;
         end;
         value b
     end;

// determines if the element does not occur elsewhere in the string
    function isUnique(x : X) : bool
    pre x in m
    ^= forall i,j :: m.dom :- (m[i] = x & m[j] = x) ==> i = j;

// counts all the unique elements
    function countUnique : nat
    ^=  #(those x :: m :- isUnique(x));

// determines if the sequence is injective (i.e. contains only
// unique elements
    function isInjective : bool
    ^= forall x::m :- isUnique(x);
```

A method that sorts the non-empty sequence is developed.

```
//sorts the non-empty sequence
    function sort : NonEmptySequence of X
        satisfy result.ranb=m.ranb,
        forall i::1..<#result :- (result[i-1]<=result[i])
        via
         var values:bag of X!=m.ranb,
             b : seq of X!=seq of X{};
       loop
            change b, values
            keep values'++b'.ranb = m.ranb,
                 forall i :: 1..<#b' :- (b'[i-1]<= b'[i])
```

```
            until #values'=0
            decrease #values';
                b!=b.append(values.min);
                values!=values.remove(values.min);
        end;
        value NonEmptySequence of X{b}
    end;
```

A selection of other operators are developed to complete the specification.

```
    operator # : nat
        ^= #m;
    operator [](i:nat) : X
        pre i<#m
        ^= m[i];

    function ranb : bag of X
        ^= m.ranb;

end;
```

### 6.3.2 Higher Order Functions

Perfect supports a few of the common *higher order functions* that are found in functional programming languages [22]. The Haskell functions `fold`, `map` and `filter` [53] are described by the Perfect functions `over`, `for` and `those`. Unfortunately, additional higher order functions cannot be created in Perfect and those that do exist cannot be extended for new data types. An attempt was made to use the language of generics to construct a generic class, `HOF` that provides a mechanism to encode and extend higher order functions in Perfect.

A class providing a template for higher order functions, allowing developers to construct their own functions, taking advantage of the higher order functions in this class is developed.

```
deferred class HOF of (X,Y)
^=
interface
    build{};

    // Higher Order functions

    nonmember function broadcast(a:class A,
                            list:seq of class B):seq of pair of (A,B)
        ^= for i::0..<#list yield pair of (A,B){a,list[i]};

    nonmember function zip(a:seq of class A,
                           b:seq of class B):seq of pair of (A,B)
        ^= for i::0..<min(#a,#b) yield pair of (A,B){a[i],b[i]};

    nonmember function mapF(f:from HOF of (class A,class B),
                            list:seq of A):seq of B
        ^= for x::list yield f.myFunction(x);

    nonmember function filter(f:from HOF of (class A,bool),
                              list:seq of A):seq of A
        ^= those i::list :- f.myFunction(i);

    nonmember function foldL(f:from HOF of (pair of (class A,A),A),
                             list:seq of A):A
        pre ~list.empty
        decrease #list
        ^= ([list.tail.empty]:
                list.head,
            []:
                f.myFunction(pair of (A,A){list.head,
                                           foldL(f,list.tail)})
           );

    nonmember function foldR(f:from HOF of (pair of (class A,A),A),
                             list:seq of A):A
        pre ~list.empty
        decrease #list
        ^= ([list.tail.empty]:
                list.head,
```

```
            []:
                f.myFunction(pair of (A,A){foldR(f,list.tail),
                                                    list.head})
            );

    nonmember function compose(f:from HOF of (X,class B),
                               g:from HOF of (B,Y),a:X):Y
        ^= g.myFunction(f.myFunction(a));

    // Deferred function that represents a single order function
    // Defined on class by class basis as required
    deferred function myFunction(x:X):Y;
end;
```

A class that declares the function `double:Y` is developed. This will allow any class that implements `double:Y` access to the higher order functions.

```
class Double of (X,Y) require X has function double:Y end^=
    inherits HOF of (X,Y)
interface
    build{} inherits HOF of (X,Y){};

    define function myFunction(a:X):Y
        ^= a.double;
end;
```

A class that implements `double` and may take advantage of the higher order functions described in the `HOF` class is developed.

```
class MyInteger ^=
abstract
    var val:int;
interface
    build{a:int}
        post val!=a;

    function val;
```

```
    function double:MyInteger
        ^= MyInteger{(2*val)};

end;
```

A class that uses the higher order function compose, to double one integer and then double the result is developed:

```
class QuadInt ^=
interface
    build{};

    function quad(a:int):int
        ^=(let f^=Double of (MyInteger,MyInteger){} as
                                    from HOF of (MyInteger,MyInteger);
           compose@HOF of (MyInteger,MyInteger)(f,f,MyInteger{a}).val
          );
end;
```

This experiment met with limited success. We have produced a loose model of higher order functions, but one lacking type extendibility (by contra-variance of the return type) and the elegance and simplicity of functional programming languages.

### 6.3.3 Observations

Perfects language of generics is powerful for the specification of abstract data types, but is severely restricted outside that scope. The constrained types class descriptor seems entirely wasteful as it does not permit the introduction of new methods. The support for higher order functions is inadequate, though with some work a moderately better solution is available. Overall, generics are a key component of Perfect Developer but one which could be better supported.

## 6.4 External Components

Software's ability to interact with external components is important as it allows software to be open to other platforms. Perfect Developer is unique in supporting a variety of languages that its software may be compiled into. However, offering such a scope of potential translations affects the richness of some of the mechanisms, such as file and exception handling. It also raises the unique challenge of developing a software GUI that interacts with compiled code.

### 6.4.1 File Handling

The file handling mechanism in Perfect Developer is primitive. It uses a global `Environment` object to capture all file operations. These operations are usually performed with respect to a `FileRef` which represents a pointer to a block of memory. Memory reads and writes must occur as byte or string accesses. To overcome this weakness, development of an advanced file handler class was attempted, and is presented below.

```
// A file handler object that access the environment, creates files,
// writes to them and reads from them.
// Incomplete and unsatisfactory
class MyFileHandler ^=
abstract
    var
        context: Environment,
        filename: string;
interface
    build{!context: Environment,!filename:string}
        post !openNew;

    schema !openNew
        post (var r:FileRef||FileError;
```

```
          let creationMode^=
                      set of FileModeType{create@FileModeType};
            context!open(filename,creationMode,r!)
          );

schema !openAppend(r!:FileRef||FileError)
    post (let creationMode^=
                    set of FileModeType{append@FileModeType};
            context!open(filename,creationMode,r!)
          );

schema !close(f:FileRef,e!:FileError)
    pre context.gIsOpen(f)
    post context!close(f,e!);

schema !write(s:string,e!:FileError)
    post (var r:FileRef||FileError!=success@FileError,
              f:FileRef,dummy:bool;
           !openAppend(r!) then
           ([r within FileRef]:
                // successful open
                f!= (r is FileRef) then
                context!print(f,s,e!) then
                    ([e = success@FileError]:
                         // successful write
                         !close(f,e!) then
                         ([e = success@FileError]:
                             // successful close
                             //********* TOTAL SUCCESS*********//
                             pass,
                          []:
                             // error closing
                             //******* OPENED + WRITTEN *******//
                             pass
                         ),
                     []:
                         // error in writing
                         //********** OPENED ***********//
                         // need to close file?
                         pass
```

```
                ),
            []:
                // error in opening - file never opened
                e!= (r is FileError)
            )
           );

schema !read(s!:string,e!:FileError)
    post (var r:FileRef||FileError!=success@FileError,
              f:FileRef,dummy:bool;
           !openAppend(r!) then
           ([r within FileRef]:
                // successful open
                f!= (r is FileRef) then
                context!readLine(f,s!,e!) then
                    ([e = success@FileError]:
                         // successful write
                         !close(f,e!) then
                         ([e = success@FileError]:
                             // successful close
                             //********* TOTAL SUCCESS*********//
                             pass,
                          []:
                             // error closing
                             //******* OPENED + WRITTEN *******//
                             pass
                         ),
                     []:
                         // error in writing
                         //********** OPENED ***********//
                         // need to close file?
                         pass
                    ),
            []:
                // error in opening - file never opened
                e!= (r is FileError)
            )
           );

schema !goToEndofFile(f:FileRef,e!:FileError)
```

```
        pre context.gIsOpen(f)
        post context!fastForward(f,e!);

    schema !goToStartofFile(f:FileRef,e!:FileError)
        pre context.gIsOpen(f)
        post context!rewind(f,e!);

    schema !find(f:FileRef,s:string,e!:FileError)
        pre context.gIsOpen(f)
        post (!goToStartofFile(f,e!) then
              ([e=success@FileError]:
                   pass,
               []:
                   pass
              )

             );
end;
```

Development of this class was abandoned after painstaking experimentation. The language provides little or no guarantees on file handling methods and no means to language of exceptions. File handling in Perfect must be interwoven throughout the software being developed. This is a huge cost, but appears to be the only way to incorporate file handling in Perfect.

### 6.4.2 Wrapper Class Safety

This examples shows how a trivial class in Perfect is mixed with a simple graphical user interface in Java. The `WorksWrapper` class sorts a sequence of characters passed to it from the wrapper class. The wrapper class, `WorksAccess` passes a value of an incorrect type to the method.

```
// A trivial class that accepts a sequence from the external world
// and sorts it. The sequence is declared to be of characters, but
```

```
// this will be changed by the Java
class WorksWrapper ^=
abstract

interface
    build{};
    schema sort(toBeSorted:seq of char,sorted!: out seq of char)
        post sorted!=toBeSorted.permndec;
end;

// End Perfect


// Simple Java Swing application front-end

package works;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import Ertsys.*;

// This is the top-level class that creates the Perfect back-end
// and passes a bad sequence object to it. Sequences are referred
// to as _eSeq.
public class WorksAccess implements ActionListener
{
    JFrame appFrame;
    JPanel appPanel;

    //adding a copy to the information/to the DB
    JTextField seqToSort;
    JButton sort;
    JButton throwSpanner;

    JLabel status;
  WorksWrapper theWorks;

    // Constructor
```

```
    public WorksAccess()
{
// Establishes the graphical user interface
// Create the back end
theWorks = new WorksWrapper();

// Create the frame and container.
appFrame = new JFrame("The Works");
appPanel = new JPanel();
appPanel.setLayout(new GridLayout(3,2));

// Add the widgets.
addWidgets();

// Add the panel to the frame.
appFrame.getContentPane().add(appPanel);

// Exit when the window is closed.
         appFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// Show the app.
appFrame.pack();
appFrame.setVisible(true);
    }

    // Create and add the widgets for app.
    private void addWidgets()
{
// Create widgets.
seqToSort = new JTextField(15);
    sort = new JButton("Sort the Sequence");
throwSpanner = new JButton("Throw Spanner");

status = new JLabel("Result", SwingConstants.LEFT);

// Listen to events from Convert button.

sort.addActionListener(this);
throwSpanner.addActionListener(this);
// Add widgets to container.
```

```
appPanel.add(new JLabel("SequenceToSort:"));
appPanel.add(seqToSort);
appPanel.add(sort);
appPanel.add(throwSpanner);
appPanel.add(status);
    }

    // Implementation of ActionListener interface.
    public void actionPerformed(ActionEvent event)
{
            if (event.getSource() == sort)
            //Correct behaviour of the class
{
  // Get data from user
String data = seqToSort.getText();

// Convert it to an _eSeq object
_eSeq toBeSorted = _eSystem._lString(data);

  // Create an empty sequence to receive the result in
_eWrapper__eAny sortedSeq = new _eWrapper__eAny();
        _eSeq temp = new _eSeq();

// perform the sort
theWorks.sort(toBeSorted,(char)0, sortedSeq, temp, (char)0);

// Covert it to a string object
        data = _eSystem._lJavaString((_eSeq)sortedSeq.value);

        // Print result
status.setText(data);
}
            else if (event.getSource() == throwSpanner)
{
  // Get data from user
String data = seqToSort.getText();

// Convert it to an _eSeq object
_eSeq toBeSorted = _eSystem._lString(data);
```

```
        // Create the spanner
_eWrapper_int spanner = new _eWrapper_int();
// Assign it a value
spanner.value = 1;
// Throw it into the works
toBeSorted = toBeSorted.append(spanner);

  // Create an empty sequence to receive the result in
_eWrapper__eAny sortedSeq = new _eWrapper__eAny();
        _eSeq temp = new _eSeq();

// perform the sort
theWorks.sort(toBeSorted,(char)0, sortedSeq, temp, (char)0);

// Covert it to a string object
        data = _eSystem._lJavaString((_eSeq)sortedSeq.value);

        // Print result
status.setText(data);
            }
else status.setText("Unknown event or return code");
}


    // main method
    public static void main(String[] args)
{
// Set the look and feel.
try
{
        UIManager.setLookAndFeel
        (UIManager.getCrossPlatformLookAndFeelClassName());
} catch(Exception e) {}

WorksAccess applicationObject = new WorksAccess();
    }
}

// End
```

As compilation occurs external to Perfect Developer, no static checks can be made on the security of the software. However, even at runtime there is no check that the values passed to the method are of the correct type. This problem arises because Perfect uses the `_eSeq` class as an ancestor to all Perfect data types.

### 6.4.3 Observations

Perfect Developer is very weak with respect to interacting with external components. File handling is inadequate for modern day software development while exception handling is not to be found at all. Attempts to rectify this with improved file handling facilities proved impossible because of the immense future development requirements this would place on software. Fundamentally, the compiled software is not type safe as it assumes all interactions with the graphical user interface will be completely correct.

## 6.5 Conclusions

In this chapter, a selection of illustrative examples with Perfect Developer have been presented. These examples highlight many of the areas where Perfect Developer was found to be lacking and provide a justification of criticisms put forth in Chapter 7.

# Chapter 7

# Perfect Developer - An Analysis

In this chapter, an analysis of Perfect Developer is presented. The analysis places emphasis on how Perfect Developer supports specification, implementation and verification of software. Each of these elements of the software development process is discussed drawing on the case studies of Chapter 5 and the illustrative examples presented in Chapter 6.

Perfect Developer is an advanced software construction tool. It incorporates elements from the formal methods community, such as verification and refinement, with solution, such as prototyping, from the industrial. As such Perfect Developer straddles two distant, yet related domains. Its goal of producing reliable software that can be proved correct should be applauded.

The Perfect language achieves its goal of being compact, easy-to-learn and expansive. It incorporates techniques for specification, such as algebraic, model-oriented and design by contract, that are rarely put together. Software that is provably free from errors can be implemented. The verifier can discharge a high percentage of common proof obligations, a reported 90%, in most software. The development process is

formalised, but never to the extent that lessens the appeal of the tool. The breadth of ideas that are brought together is vast and well thought out, yet their execution is poor. An analysis of the tool will now be presented.

## 7.1 Specification

Perfect Developer provides a rich specification language that is composed of many novel features. The language is easy to learn while providing a wide range of specification components. There is usually at least one way, if not more, to specify what is required. Perfect provides an advanced language of generics as well as supporting a selection of higher order functions. In this section, the support Perfect Developer provides for specification is critiqued.

### 7.1.1 Specification Styles

Perfect includes multiple specification styles within a single software development framework. The Library Database case study (See section 5.1.2), included an algebraic property that defined the borrowing and returning of a library item as having no net effect to the library database. This specification would not have been available to a purely model-oriented or design by contract specification language. The richness of specification styles results in a steeper learning curve than is necessary.

Inclusion of a variety of specification styles helps to promote interest in Perfect Developer among different software development communities. In learning Perfect Developer, students found the mix of techniques confusing as it is unclear where one

specification style is superior to another. The change of mindset that is repeatedly required to write algebraic, model-oriented and design by contract specifications during a single development phase is costly. Including a selection of styles within a specification creates problems for the verifier. In the Library Database case study, the verifier failed to verify the individuality of library items this uniqueness was defined as a property rather than an invariant. While employing a variety of specification styles in Perfect Developer is beneficial, the associated cost to software development is quite high.

### 7.1.2 Generics

Perfect provides an advanced language of generics in which to construct classes, in particular abstract data types like hash-tables (See Appendix D.1). Unfortunately, the notation used to describe this form of parametric polymorphism is cumbersome for the developer. The `requires` clause can grow arbitrarily large with each required function potentially having pre-condition and post-assertion definitions in the header as observed in the higher order function example (See 6.3.2).

Perfect overcomes the insecurities that exist in the type system of Java [42] and C++ [16], but the resulting specification is not elegant. As larger software is constructed, generic classes become more complex and harder to understand. Generics in Perfect do not provide adequate support for specification, but do indicate a big advancement over previous attempts such as with Java and C++.

### 7.1.3 Higher Order Functions

Perfect is not unique among object-oriented languages in providing features from the functional programming paradigm [88, 11, 79]. However, Perfects higher order functions are not as rich a functional component as these other languages. The three operations of `fold`, `map` and `filter` are described in Perfect through the functions `over`, `for` and `those` respectively. In addition to the limited number of higher order functions available, the application of these functions are restricted to the basic collection data types.

The inclusion of higher order functions provides a richness and elegance to specification with Perfect. However, developers are provided with no means to construct unique higher order functions nor to extend these functions to custom abstract data types. We have seen an attempt to develop the language of higher order functions (See 6.3.2), and while some success was achieved, the result was restricted and lacks elegance.

### 7.1.4 Concurrency

Perfect Developer makes no attempt to cater for concurrent behaviour or temporal specifications. This is perhaps the greatest limitation of Perfect as a specification language. The domain of reactive software, as illustrated by the Resource Manager case study (See 5.2), is beyond the scope of Perfect to specify.

The design decision not to include concurrency in Perfect Developer makes the tool less complex than it could be. However, the impact of this decision is that many of the great challenges faced by software developers are left unaddressed by the tool.

Deadlock, livelock and race conditions don't arise in software because concurrent behaviour is not permitted rather than because a verification is provided. This solution ignores modern software advancement and removes the ability to develop powerful and interesting reactive software projects.

## 7.2 Implementation

Perfect Developer is somewhat unique within software development tools in that it offers implementations to be constructed automatically from specifications. The ability to prototype software so quickly gives developers the chance to validate their software early in the development process. The inclusion of refinement as a mechanism to create better, more efficient software must be applauded. While refinement offers the developer the chance to write highly efficient software, any efficiency gain is marginal in comparison to the cost of employing value semantics. The implementation language suffers because it has inadequate file handling and no exception handling or graphic capabilities, which results in the need for wrapper classes. These criticisms will be discussed in greater detail in this section.

### 7.2.1 Refinement

Refinement in Perfect Developer affords developers the chance to specify software correctly before spending time on constructing an equivalent correct implementation. The support that Perfect Developer provides for refinement can be seen in great detail in [28], but even in these small examples the tool is inadequate. Worse is the problems that arise on medium scale case studies such as the refinement in the Library

Database case study (See 5.1.7). Here a refinement of a set object to a hashed-bucket was attempted, but verification was impossible. Also, the refinement step produced a source file that was unnecessarily large.

The syntax of refinement in Perfect is poorly constructed. The developer must construct a retrieve function from implementation to specification during data refinement. This constrains data refinements where a more general retrieve relation would be required. When refinement occurs, specification and implementation code becomes intermixed within the one source file. Often, the syntax of a specification must change after refinement. For example, the semi-colon that follows method specifications must be moved to follow the implementation after algorithm refinement. This makes the source code non-uniform and excessively long. Perfect Developer has not included refinement correctly, even though its inclusion at all is a major step in the right direction.

### 7.2.2 Value Semantics

Perfect Developer overcomes the challenge of constructing provably correct software by employing value semantics. Value semantics greatly simplifies the task of full verification, removing the possible occurrence of aliasing from software. Unfortunately, value semantics is a costly technique to employ when implementing software. This is particularly true of object-oriented software, where assignment involves the destruction of the previous value of an attribute.

The cost arises when software is described using functions rather than schemas. For example, consider the function `append(a:X)` on sets that returns a new set with the item `a` inserted. Note that a new set is returned. If the set contains 1000 objects,

the set must be cloned, and in order that no element is aliased, each element of the set must also be cloned. The cost of merely inserting an item to a set is enormous. From discussions with the architects of Perfect Developer, it became apparent that an internal tree structure is used to overcome cost for this particular example, but some cloning is still necessary.

The problem does make an impact, however, with user defined classes. Abstract data types, like the Hashed-Bucket class in Appendix D.1 , will have to clone data in order to ensure correctness. Complex objects that are composed of other complex objects require deep cloning to prevent aliasing. No estimate of the amount of unnecessary cloning has been made, but the architects stated that at least 40% of the cloning that occurs could be removed. There has been no official statement that this situation has been rectified at the time of writing.

### 7.2.3 File and Exception Handling

Perfect Developer supports a very primitive form of file handling, as we observed in the Library Database case study (See 5.1.3 and 6.4.1). When errors arise, the software signals an error, but very little information about the cause can be accessed. File handling is performed through a "global" object that must be passed around the software. Developers should be able to treat files as just another abstract data type, only one that offers the advantage of permanent storage. File handling is no doubt complex, but it seems Perfect Developer chooses to ignore the problem rather than provide an adequate solution.

Worse still is the complete lack of exception handling in Perfect Developer. Exceptions in software do not necessarily signify a lack of correctness, as it is commonly

believed, but instead signify that some events are exceptional and require unique treatment. Perfect Developer does not accept any event as being an "exception". Any exceptional circumstances that arises during execution of software constructed with Perfect Developer forces termination of the software. Worse still is the case when an exception occurs, and the software is allowed to continue even though it is in an incorrect state. (see Appendix G).

### 7.2.4 Wrapper Classes

Perfect Developer is primarily designed for the development of rigorous "back-end" components that will be utilized as part of larger software. As a result of this, no thought went into the development of graphical user interfaces in Perfect. The failure to include a library of graphic operations forces the developer to construct the "front-end" in the language chosen for compilation, and integrate this code with the compiled Perfect code. However, this is neither easy nor clean because method headers are often mangled during compilation. It is clear, from the compiled Library Database files (see Appendix H), that the Java code produced is overly complicated and difficult to access. The developer must understand this code, at a major expense to development time, before the GUI can be constructed.

Another difficulty that faces the developer is that Perfect does not use the primitive types of the target language to implement the Perfect primitive types. If an integer value is required as a parameter, an `_eInt` object must be instantiated. Similarly, strings must be converted to the `_eSeq` type which may result in a lack of type safety (See 6.4.2). These translations are not documented as part of the reference manual, but can only be found through experimenting with a provided example and

a web-page describing the process.

## 7.3 Verification

Perfect Developer offers an automated theorem prover with which to perform verification. This helps to overcome the high expertise requirements faced by many of the other software development tools that were presented in Chapter 3. The tool provides a single interface to verification, while the verifier produces proof output in the form of html, text or TEX. However, verification with Perfect Developer is not without challenges. The tool suffers somewhat from the automation process. The verifier is severely limited in what it can achieve when presented with complicated proof obligations. This is made worse by the lack of adequate documentation of the verifier component regarding its architecture and implementation. In this section, we critique the support for verification provided by Perfect Developer.

### 7.3.1 Automated Process

Perfect Developer supports a fully automated verification process. When a software specification is constructed, it is presented in its entirety to the verifier which attempts to establish its correctness. Unfortunately, the verifier treats the specification as monolithic to ensure all specification changes are included in its verification. This means that components that have remained unchanged need to be re-verified during each verification attempt. As a result the time to verify a large piece of software is immense. The completed Library Database case study, for example, takes several hours to be verified.

An automated, rather than interactive, theorem prover is employed by Perfect Developer to encourage developers who want verification without the high cost usually associated with formal methods. While this is a noble goal, it is one fairly difficult to achieve. Developers need to have some basic understanding of the verification process, ideally with the option of interacting with the theorem prover, in order to verify complex software. Perfect Developer prefers to force developers to study the proof output, which is difficult for the average developer to parse, in order to discover mistakes in the software.

It should be noted that in the latest version of Perfect Developer, not available at the time of writing, a "suggestion" feature has been incorporated that attempts to pinpoint the cause of verification failure.

### 7.3.2 Documentation

There is very little known about the architecture of the verifier component of Perfect Developer. There is no supporting documentation of the strategy or tactics employed during the verification let alone its limitations. There is no proof that the verifier is either correct or sound. All that is publicly known is that the verifier uses a custom built theorem prover. A complete formal description of the verifier must be provided before Perfect Developer can be really considered for research or industrial applications.

Similarly, there is no documentation of the language used by the verifier to present proofs and failures to prove. While it is usually trivial to follow each step of a proof generated by the verifier, this is not so easy when the verifier doesn't find a proof. As the verifier makes several attempts to find a proof, presumably generating deep

and complicated proof trees, a complete output of the attempt would be impossible to parse and of little value. However, the output that is provided usually makes reference to variable names not included as part of the original specification. These kinds of output can assist the developer, but only after much time is spent trying to understand them.

### 7.3.3 Limitations

Many limitations in the verifier were found through the experiments presented in the case studies and examples. This thesis can not successfully categorize most of these errors because there is no documentation on the architecture of Perfect Developer and therefore assurances cannot be made as to the causes of failed verification attempts. However, we now present three major limitations found with verification.

The theorem prover component of the verifier is not inductive, i.e. it cannot carry out inductive proofs. This fact can usually be hidden by the verifier when sensible limits are placed on proofs. Consider the trivial obligation:

```
assert forall i::nat :- i >= 1 => 2*1 >= 2;
```

This assertion cannot be proved because the standard boredom limit is reached and the verifier abandons its search. No increase in the boredom limit proved effective in producing a proof as the domain is infinite. A partial proof can be found by rephrasing the assertion as:

```
assert forall i:(1..10000) :- i>0 => 2*1 > 1;
```

In both attempts, the verifier proves the base case (`i=1`) and then proves each subsequent case. It is believed that for all assertions of this category, a similar exhaustive

search strategy is employed.

Related to this is the limitation that memory usually requires maximum and minimum integer values. It is well known that there must exist a minimum and a maximum value for implementable `int` values in real world software. Often, this upper limit is ignored in specification languages because there is no upper limit to integers in theory. However, when specification and implementation languages are combined, some consideration must be given to this problem or else the addition of two integers could result in an incorrect result. Perfect Developer does not consider this problem at all, to the extent that it is possible to write incorrect software with Perfect Developer that is verified correctly.

Perfect Developer permits the inclusion of *axioms* that assert absolute truth. The purpose of an axiom is to assist verification in those cases where failure to verify arises out of insufficient rules in the verifier. An axiom is defined by declaring the `axiom` keyword followed by a regular assertion. The assertion will never be tested for truth, just assumed to be true. However, in any attempt made to include an axiom to assist verification, the verifier appeared to ignore the axiom proceeding to fail for exactly the same reason as before inclusion of the axiom. It cannot be concluded that axioms serve no purpose in verification, as there exists no documentation regarding the tactics of the verifier, but no evidence of axioms working has been found during extensive research.

## 7.4 Conclusions

Perfect Developer should be recognised as an important software development tool. The ideas endorsed by the tool represent the best possible mix for a software development tool. The promotion of specification in advance of programming is to be applauded. The fundamentals of Perfect Developer are far above par, but most of our criticisms arose with the details. The Perfect language is too restrictive in its treatment of generics and higher order functions, yet too open in its treatment of specification styles. The compiler produces potentially highly inefficient code that cannot adequately interact with other external entities. The verifier is poorly documented and fails too often without reasonable cause or clear explanation. Perfect Developer is a good first step, but it has a long way to go before being Perfect.

# Chapter 8

# Recommendations for Improved Tool Support

In this chapter, recommendations are presented for the development of a software construction tool, referred to as SCALE. SCALE (Software Construction Assistant & Language Environment) is a software construction tool which will be designed in the mould of Perfect Developer. Here, recommendations for the SCALE tool are presented. These recommendations are generated from our critical analysis of PD as presented in Ch 7.

## 8.1 SCALE

x SCALE should present to the developer an environment in which formal software construction methods are advocated and supported throughout the development process. Software construction will be supported by the tool from the requirements through to the final implementation, relying upon an advanced refinement mechanism

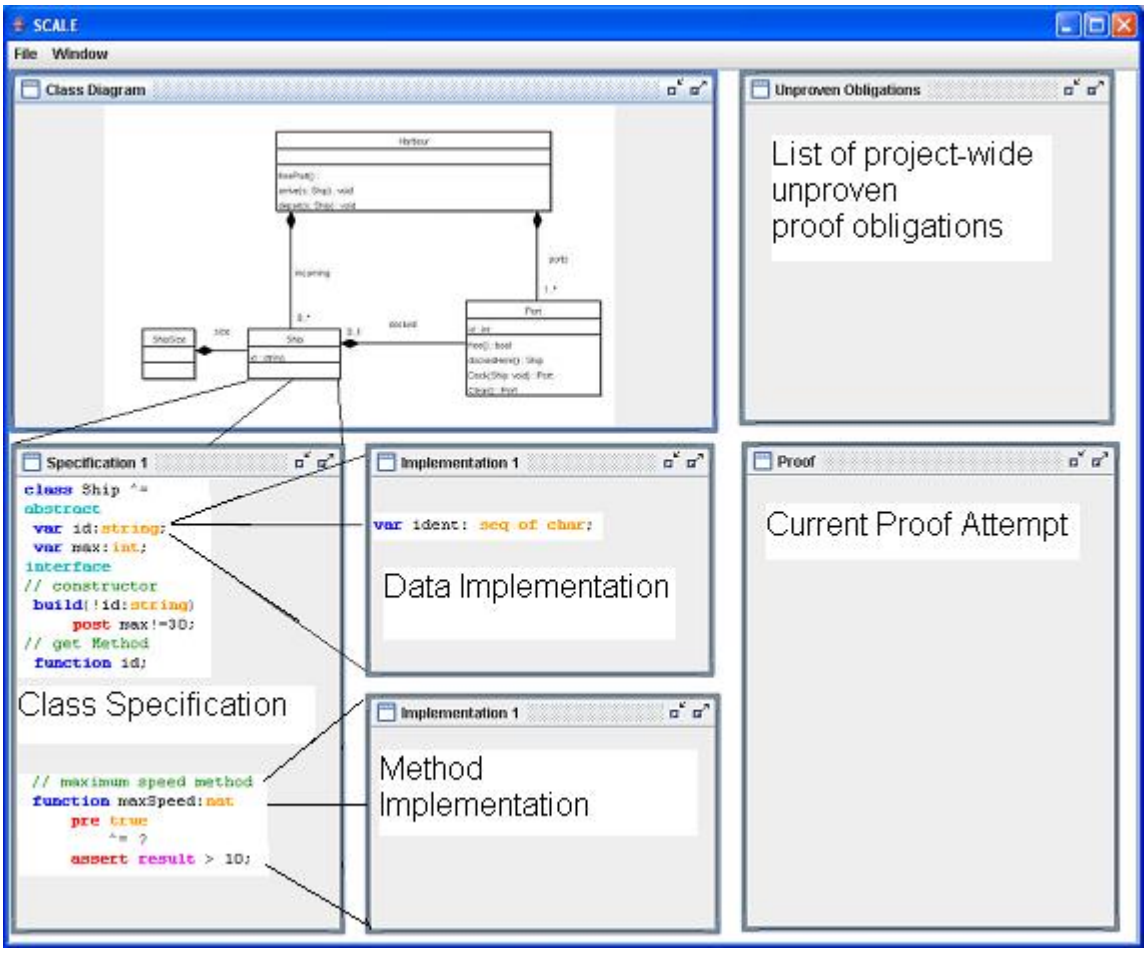


Figure 8.1: SCALE GUI - Prototype version

throughout. Specification will be made initially with UML class diagrams, but further developed with a Perfect-like language. Implementation of software will proceed from specifications directly, integrating the specification language with the target language. Verification will occur in modules, where each proof-attempt may be automated or interactive, but proofs re-used when change occurs. The major components of SCALE is presented in this section.

### 8.1.1 Project Environment

SCALE will consist of four distinct software encoding levels: requirement; class diagram; specification; and implementation. Natural language requirements will be initially defined within a text file. UML class diagrams will be developed to describe the structure of the software. Specifications of the software will be defined as model-oriented specifications augmented with design by contract assertions. Implementations will be developed as modular elements of the class specification. SCALE will support iterative refinements to be made at each level of the software description. The environment will also handle interaction with the compiler and the verifier.

Within the requirements file, each requirement will be manually associated with a *requirement symbol*. This will be used to ensure no requirement is accidentally lost during the development process. The requirements document may be refined into another requirements document that includes more detailed requirements or decomposes a single requirement into multiple requirements.

Requirements may be given structure by refining them into a UML class diagram. Those requirements that are ensured solely by the structure of the class diagram will be indicated by manually. This indication will be made by marking the associated requirement symbol into the diagram with an annotated justification of why the refinement is acceptable. The interface will allow the class diagram to be exported to and imported from xmi format. Class diagrams will be kept up to date with changes in the specification and the implementation as it will provide the only mechanism for class and feature introduction. Global specifications may be included in this diagram if appropriate.

### 8.1.2 Language

The language for use in development with SCALE should be similar to Perfect. It should consist primarily of a specification language that has an implementable subset. It will exhibit a single paradigm of object-oriented programming and the specification elements will be tied to that paradigm. Learning the language will be simplified by keeping the theory compact by not supporting algebraic specifications. Instead a rich language of generics and higher order functions will be provided to assist specification.

The specification language should address the issue of inheritance and polymorphism by recognizing at least three inheritance relationships: *is-a*; *extension*; and *adaptation*. The is-a relationship will be used when sub-classing by specialisation is permitted. The extension relationship will be used when strict sub-typing is required. The adaptation relationship will be used when the child class may change the parent class specification. Each inheritance relationship will include a thorough treatment of binary methods and dynamic binding.

### 8.1.3 Compiler

The compiler should translate the software into C# and Java. This would promote using the tool with language independent frameworks like .Net and platform independent frameworks like J2EE. A de-compiler, which extracts the headers, but not the implementation, of C# and Java classes should also exist to promote re-use. This would support the integration and re-use of external components within the tool. More details of this can be found in 8.3.2.

### 8.1.4 Verifier

The verifier will be built upon an interactive theorem prover to provide developers with expertise the freedom to undertake proofs manually. These proofs will be composed within the Proof Attempt window. Proofs completed will remove the proof obligation from a list of project-wide proof obligations. Changes made to the software may result in the re-introduction of proof attempts. This will require SCALE to be aware of software changes and the impact on proof obligations.

## 8.2 Specification

In this section, recommendations for the specification language used by SCALE are presented. They enhance the support for specification that is currently provided by Perfect Developer. The language of generics should be made increasingly expressive by include a language of *kinds*. An improved mechanism for higher order function creation and application is discussed. A primitive set of temporal specification operators is suggested to introduce concurrency in SCALE, a feature lacking in Perfect Developer.

### 8.2.1 Kinds

Programming languages usually only distinguish between values and types, (objects and classes respectively). Cardelli [25] suggested extending this to include *kinds*, or the *"type"* of types [62]. The inclusion of a third level of kinds would provide a means to describe increasingly abstract properties of a program. The notion of kinds is implicitly included in programming languages where a kind `Type` exists, describing

the kind of all types (in Java this is achieved ad hoc via the `Object` class, or in Perfect via the `any` class).

An inclusion of kinds would be truly beneficial for languages that permit complex polymorphism or generics to be described. Many of the difficulties that faced the Generic Java project [42] could be overcome by employing kinds to describe parametric polymorphism. For example, in Kind C++ [108], a language that incorporates kinds with C++, software could be developed with greater clarity than could be achieved by template classes in C++ alone. Kinds provides a unique way to treat parametric polymorphism, but one which should be intuitive to software developers as it is analogous to typing.

Kinds could improve the language of generics advocated by Perfect. Specification of generic classes would be easier resulting in cleaner and more meaningful data structures. The safety of polymorphism would be much higher as classes, and methods, could be described with respect to their holding certain properties such as function associativity, commutativity or object equality. The language of SCALE would be more elegant and expressive with only a small additional cost to its theory.

SCALE would require a language of kinds, similar to that used to describe classes, to be developed. Here a kind could be defined by a description of its mathematical properties or through composition of other kinds. The assertion language of Perfect is sufficiently rich and would serve as the initial basis for SCALE. Kinds would greatly increase the support SCALE could provide for specification in terms of richness and clarity.

### 8.2.2 Higher Order Functions

Higher order functions provide developers with a means of describing functions at higher levels of abstraction, encouraging re-use of software and elegance of specification. We have also observed their inclusion, albeit a limited inclusion, in Perfect as well as attempts undertaken to increase this support (See 6.3.2). Their value is clear when reading software constructed in a functional language like Haskell [101] or ML [104].

It is recommended that a language for the description and construction of higher order functions be included in SCALE. The language would be relatively simple to include if built with the assistance of kinds that was discussed previously. Higher order functions could be defined across kinds, thereby automatically available to developers when a class is defined to be of a particular kind. Unique higher order functions could be defined on types if kinding does not provide the required richness.

Including a language of higher order functions, as attempted in 6.3.2, would benefit SCALE as a specification language. The theory behind SCALE would be kept neat but would be more expressive than Perfect, which only supports a limited form of higher order functions. This language could proceed directly from the language of kinds also advocated by the thesis. This would further enhance the elegance of SCALE and of the software constructed with its support.

### 8.2.3 Temporal Specification

One of the strongest criticisms made about Perfect is with regard to the lack of a temporal specification language (See 5.2.2 and 6.1.4). A specification language that describes reactive and concurrent behaviour is becoming necessary as modern

software becomes increasingly dependent upon this behaviour. It is proposed that a set of temporal operators should be included in SCALE in some format. Ideally, a CSP-like [91] language would be included.

However, unlike other approaches that attempted to unite temporal and model oriented specifications [93], something that resembles CSP||B [92] is preferred. Here separate specifications are developed, one defining the CSP controller and some message passing mechanisms, the other defining B machines that describe the data model. This is a powerful and safe combination of techniques.

The inclusion of a language for temporal specification would greatly expand the domain of problems where SCALE could be applied. This language would be easily included by means of a parallel CSP module for the specification of concurrent and temporal behaviour. Single threaded software, described in the model-oriented and design by contract specification, could still be easily developed. More complex behavioural specifications could be defined in parallel in SCALE.

## 8.3 Implementation

In this section, the implementation language limitations of Perfect Developer are addressed by SCALE. Most importantly, SCALE should provide a language that describes exceptions. Better support for file handling and wrapper classes could be achieved by increasing the ability for external components to be supported. The worse element of implementing software with Perfect Developer arose because of the choice of value semantics in Perfect. It is believed that through clever treatment of the *frame condition* of a method, the aliasing problem would be avoided in software developed with SCALE.

### 8.3.1 Exception Handling

Real-world software often results in exceptional circumstances occurring. The lack of any exception handling in Perfect Developer is a severe limitation. It is recommended that SCALE should include a safe exception handling mechanism that does not force termination immediately in an exceptional situation.

Eiffel provides an advanced exception handling mechanism that enables software to try to correct errors and then continue execution. When an exception is thrown in Eiffel, a method can treat the exception individually based on its type. A `rescue` method is then executed whose responsibility is to mend the invariants broken during the exception. It then calls `retry` which re-calls the method with the original parameters. Only if this attempt also fails does the software force termination.

Rigorous software that only fails when a catastrophic error has occurred can be produced by incorporating this language of exception handling into SCALE. This will overcome some limitations met by Perfect Developer resulting in an improved tool that constructs more reliable software.

### 8.3.2 External Component Support

The overhead of including a graphics library in a new software language is one most developers wish to bypass. In Perfect Developer, this resulted in the need for wrapper classes that perform the graphical front-end operations of the class. However, integration was necessary and difficult. It is recommended that SCALE include a language in which external components may be specified and used by developers. This external component support would enable file handling issues to be overcome.

It is assumed that software developed with SCALE may be compiled into Java

code similar to Perfect Developer. A Java import feature could be included that accepts Java code and creates a skeleton specification of the Java code. This skeleton would include all the methods available. The code could be further specified using algebraic properties or design by contract specifications. These specifications could not be verified for the project, but mistakes with the software could be isolated to an incorrect understanding of the imported Java code. This will provide SCALE software with the full range of Java graphics libraries and file handling support for very small cost.

The external component support could be further extended to include an integration with the .Net platform, which would permit the development of inter-language applications. It is believed that the construction of such a component will provide SCALE with a strong link to current software development practice. This encourages developers to use the tool to construct components correctly with easy integration to other elements of the software. It could also allow piecemeal introduction of SCALE software into existing software systems.

### 8.3.3 Frame Conditions

The most inefficient element of software constructed with Perfect is the cost of object cloning that arises as a result of value semantics. Software should be able to utilize reference semantics without compromising safety because of aliasing. One of the most successful techniques for this is achieved by *frame conditions* [68]. A frame of a method describes the set of objects that may change state during the execution of the method. The method can be proved to alter only those objects defined by the frame. Subsequently, a theorem prover can be used to discover variables that may be

aliases appearing in the frame.

Krakatoa [67] is a software tool that can reason about Java programs annotated with JML checking frame conditions and discovering aliasing. Unlike the other tools we have discussed (See Chapter 3), Krakatoa treats variable locations symbolically rather than their object values. It discovers aliases by simulating software with a model of the memory access, assignment and allocation protocols of Java. This can prove the absence of aliases in software without introducing the high cost associated with implementing value semantics.

It must be noted, that employing the technique used by Krakatoa severely limits the ability for SCALE to develop platform independent software. Software that was verified correct with respect to one memory management protocol may fail on another. However, frame conditions can be used to increase the efficiency of an implementation while providing a strong, if not total, guarantee of run-time safety. Offering the option of value semantics for platform independence and complete correctness is justified, but can be improved with additional work.

## 8.4 Verification

The verifier, a crucial element of Perfect Developer, was deeply criticised in Chapter 7 because of the lack of information about its architecture and its operation. The theorem proving engine of SCALE is an extremely important design decision. While complete automation is seen as a disadvantage to full verification, it would be of benefit to include some level of automation. This could be achieved through the development of *tactlets*, small proof strategies. These tactlets could evolve to solve large and complex verification conditions with or without developer interaction.

### 8.4.1 Theorem Prover

Verification proved to be a chore with Perfect Developer time and again as a result of the weaknesses of the verifier. A concern of this thesis is the lack of sufficient documentation regarding the technology that is behind the verifier component of Perfect Developer. This concern could be easily dismissed in SCALE by using a pre-existing theorem prover to perform the verification of software. There exists a host of extremely evolved and well-documented theorem provers.

HOL [43] is a theorem prover that can reason about propositions written in higher order logic. Coq [90] is a proof assistant built on the framework of the *Calculus of Inductive Constructions*. PVS [82] is an interactive rewrite theorem prover built upon a sequent calculus. Isabelle [105] is a highly flexible theorem prover designed expressly for software verification that accepts a variety of formal calculi. All of these theorem provers are formally described and documented to the highest level. Any would be a good choice for incorporation with SCALE.

Developers could be provided with excellent documentation of the strengths and weaknesses of the verifier if any of these theorem provers acted as the engine for verification of software. The developer would also be provided with the means to manually solve difficult proof obligations. The verification process itself would become more involved and less costly requiring the verification of repeatedly proving obligations.

### 8.4.2 Tactlets

Using an interactive theorem prover had the unfortunate side-effect of increasing the expertise requirements in using SCALE. It is suggested that this limitation be overcome by allowing *tactlets* [17] to be described. A tactlet describes a proof strategy

that a theorem prover may use to dispatch an obligation. The tactlet may be arbitrarily complex, but are traditionally quite simple. They have been primarily used in manual theorem provers to assist verification.

It is recommended that a language be defined to include tactlets that can be described by composing existing primitive tactlets. This language would be available to developers who wish to prove software correct manually. Successful tactlets could be shared, after a validation process, with other developers, thereby increasing the power of the verifier. In this way, a customized verifier could be constructed where the developer may select those tactlets most appropriate to the task of verifying common software problems. This technique would support evolution of the verifier, while allowing developers to choose their own level of interaction with the theorem prover.

Tactlets provide a way to support verification at all ends of the expertise scope. Those developers with limited understanding of the mathematical principles can rely on a fully automated, though extremely time consuming process, whereas developers with high expertise can create proof strategies that can be shared to improve the tool. Clearly, much effort would be needed to ensure the theorem prover is never provided with inconsistent or incompatible tactlets. This cost can be reduced by requiring tactlets to be accompanied by manual proofs and only highly valued tactlets be included in official releases.

## 8.5 Conclusion

In this chapter we have presented a set of recommendations that could produce a powerful software development tool. These recommendations solve the problems that

were encountered with the Perfect Developer software tool. They combine the best elements of Perfect Developer with new, emergent technologies. In providing these recommendations, the thesis has achieved its goals and will shortly conclude.

# Chapter 9

# Conclusions

In this chapter, some conclusions about the thesis are presented. An overview of our experience with Perfect Developer is presented, evaluating it with respect to its support for specification, implementation and verification. The recommendations presented in Chapter 8 are reviewed with respect to their practicality, usefulness and limitations. The thesis concludes by pointing at future work which may be carried on from the work, indicating how recommendations for an improved software development support tool could be realised.

## 9.1 Perfect Developer

Perfect Developer has been presented, illustrated and analysed in this thesis. We have seen how it compares favourably with other support tools. We experimented with the tool on medium and small scale software development projects, illustrating many interesting and some novel techniques. Our analysis found the tool to be a good first step for automating formal software development, but one that has a long way to go

before it can be considered industrial strength. We now provide an overview of our observations on how the tool supports specification, implementation and verification.

### 9.1.1 Specification

Specification is highly supported by Perfect Developer. In most cases, Perfect was found to be expressive enough to describe the software that was developed. If anything, the language is too rich, supporting a range of specification styles making the theoretical basis of Perfect Developer unnecessarily complex. Better support for generics and higher order functions would benefit the language, but workarounds can usually be found. The greatest weakness of the specification language was its lack of support for concurrency and temporal operators. It is felt this is a much needed element of any support tool, yet Perfect Developer does not include it.

### 9.1.2 Implementation

Our experience with implementing software using Perfect Developer was generally disappointing. While Perfect supports the refinement of specifications into implementations, this occurs too quickly and verification is extremely difficult. The treatment of value semantics is the greatest obstacle, that results in an enormous amount of unnecessary cloning. The lack of adequate file and exception handling is also regrettable. Hence we believe that implementation could be much better supported by Perfect Developer.

### 9.1.3 Verification

The verifier was the dominant weakest in Perfect Developer. The lack of documentation leaves the developer with no means of learning about failed verification attempts other than through trial and error. The proof output is excessively long-winded when a proof is formulated and entirely obscured when a proof fails. The process itself is costly of time, requiring a monolithic verification of all the verification conditions each iteration through the verification. The verifier can prove only simple obligation, and specifications must be spoon fed to the verifier if complete verification is ones goal.

## 9.2 Future Work

The next essential step for this work is an implementation of the SCALE tool. This implementation is conceived as the next phase of TOOLAP, a joint research project between National University of Ireland, Maynooth and Dublin City University. TOOLAP aims to develop a method for constructing reliable and trustworthy object-oriented software that integrates theory and practice. The practical component will consist of a software tool, SCALE, that supports the programmer in constructing specifications, in testing them for consistency and correct, efficient behaviour. The recommendations that have been made in this thesis will now be evaluated to consider their practicality and usefulness.

SCALE is a highly practical project that would build upon the successes of Perfect Developer while overcoming the limitations of the tool. As Perfect Developer is implemented, it can be assumed that most of the elements of SCALE are practical.

The inclusion of stepwise, iterative refinement would come at only a syntax cost. Kinds and higher order functions could be included that take advantage of concepts in the object-oriented language for a cleaner presentation. Including a language of temporal operators may come at a high cost, however, treating the language as a parallel element of SCALE rather than integrating it with the language simplifies this problem.

The implementation language advocated here would also be easy to include, based on the evidence of other object-oriented programming languages. Using a rich hierarchy of `Exception` classes, a simple-to-use exception handling mechanism could be constructed over time. The support for external components is costly, requiring a compiler for Java and C# be developed, but it is not too challenging. The reverse engineering component, that translates Java or C# is a much easier task requiring only that class and method names be extracted. While including frame conditions in the implementation language is relatively easy, providing a verification is a much greater challenge. The work undertaken on Krakatoa suggests it is possible, but this shows a potential weak point of the tool.

Integration with a verifier of choice is a significant issue. While many choices do exist, selecting one that best suits the problems faced by software construction in general is an open-ended issue. Overall, SCALE is practical to construct and beneficial to software construction. It would overcome the limitations of Perfect Developer, and provide better support for the construction of the case studies presented. Its support of formal methods in software construction would ensure a good comparison with other support tools. SCALE would assist formal software construction, and thereby improve software quality and reliability.

# Appendix A

# Harbour Software

Here is the Harbour software definition.

## A.1 BrokenShip.pd

```
class BrokenShip ^= inherits Ship
  interface
   build{n:string,s:ShipSize,m:nat} inherits Ship{n,s,m};
   absurd function maxSpeed:nat;
end;
```

## A.2 FastShip.pd

```
class FastShip ^= inherits Ship
  interface
   build{n:string,s:ShipSize,m:nat} inherits Ship{n,s,m};

   redefine function maxSpeed:nat
     ^= max
    assert result > 50;
end;
```

## A.3 Harbour.pd

```
class Harbour ^=
abstract        // Specification of model
 var ports:set of Port,     // all ports available
     incoming:Queue of Ship;  // incoming queue of ships

 invariant #ports >0;       // a harbour has ports

 function freePorts:set of Port   // the ports available
   ^= those p::ports :- p.free
```

```
   via value vacantPorts            // refined
   end;

 function haveFreePorts:bool      // are there free ports?
   ^= #freePorts >0
   via value #vacantPorts>0         // refined
   end;
 function hasQueued:bool          // are there ships queued?
   ^= ~incoming.empty
   via value ~incoming.empty      // refined
   end;
 // if we have free ports, we don't have ships queueing
 invariant haveFreePorts ==> ~hasQueued;

internal            // Data Refinement

 var vacantPorts:set of Port,     // free ports
     usedPorts:set of Port;       // ports in use
 function ports                   // retrieve function
   ^= vacantPorts++usedPorts;

 // No port is free and in use
 invariant vacantPorts**usedPorts=set of Port{};

 // open ports are free & closed ports are not
 invariant forall p::vacantPorts :- p.free;
 invariant forall p::usedPorts :- ~p.free;
confined                         // Private methods

 // a ship lands at the given port
 schema !land(s:Ship,b:Port)
   pre b in freePorts       // the port is free to begin
   post ports!=ports.remove(b) then     // no longer free
        ports!=ports.append(b.Dock(s))  // ship docked
   via
       vacantPorts!=vacantPorts.remove(b),    // refined
       usedPorts!=usedPorts.append(b.Dock(s))
   end;
 // a queued ship lands at the freed port
 schema !landQ(p:Port)
   pre haveFreePorts       // we have a free port
   post (let s^=incoming.head;      // first ship on the queue
         incoming!=incoming.tail then // remove him
         !land(s,p)                  // land the ship
        )
   via (let s^=incoming.head;       // refined
         incoming!=incoming.tail then
         !land(s,p)
        )
   end;
interface                   // Class methods
 operator =(arg);           // Refined =

 build{n:nat}     // Constructor
    pre n>0       // there will be positive number of ports
    post ports!=(for i::1..n yield Port{i}).ran, // make ports
         incoming!=Queue of Ship{}                // no ships queueing
    via          // refined
        vacantPorts!=(for i::1..n yield Port{i}).ran, // all ports free
        usedPorts!=set of Port{},              // no ports closed
        incoming!=Queue of Ship{}                //no
```

```
   end;

// output
redefine function toString:string
  ^= "Incoming : " ++ incoming.toString ++ " & ports :" ++ ports.toString
  via value "Incoming : " ++ incoming.toString ++ " & ports :"
                                          ++ (vacantPorts++usedPorts).toString
  end;

// from Abstract section
function freePorts;
function haveFreePorts;
function hasQueued;

// Get the least used port (lowest id)
function freePort:Port
  ^= freePorts.min
  via value vacantPorts.min
  end;

// is s docked in our harbour?
function isDocked(s:Ship):bool
  ^= (let shipsDocked^= for b::ports yield b.dockedHere;
      s in shipsDocked
     )
  via let shipsDocked^= for b::usedPorts yield b.dockedHere;
      value s in shipsDocked;
  end;

// where is s docked in our harbour?
function myPort(s:Ship):Port
  pre isDocked(s)
  ^= that b::ports :- b.dockedHere=s
  via value that b::usedPorts :- b.dockedHere=s // refined
  end;

// a ship arrives, land the ship or queue her
schema !arrive(s:Ship)
  post ([haveFreePorts]:!land(s,freePort),  // if free ports, land...
        []:incoming!=incoming.append(s)     // ...else add to queue
       )
  via ([haveFreePorts]:!land(s,freePort),  // refined
        []:incoming!=incoming.append(s)
       )
  end;

// a ship leaves, free Port and check queue for new ships
schema !depart(s:Ship)
  pre isDocked(s)
  post (let dockedPort^= myPort(s);
        let emptyPort^=dockedPort.Clear;
        ports!=ports.remove(dockedPort) then  //no longer docked
        ports!=ports.append(emptyPort) then   //now empty
        ([hasQueued]: !landQ(emptyPort),      // land queued ships or..
         []:pass                              // ..do nothing
        )
       )
  via let dockedPort^= myPort(s);             // refined
      let emptyPort^=dockedPort.Clear;
      usedPorts!=usedPorts.remove(dockedPort);
      vacantPorts!=vacantPorts.append(emptyPort);
```

```
       if [hasQueued]: !landQ(emptyPort);
        []:pass;
       fi
   end;
end;
```

## A.4 Port.pd

```
class Port ^=
abstract        // Specification of model
 var id:nat,                 // Port id
     docked:Ship||void,      // a docked Ship or nothing
     used:nat;               // used so many times

confined                         // Private methods

// constructor which initializes all attributes
 build{portId:nat,isDocked:Ship||void,u:nat}
     post id!=portId,
          docked!=isDocked,
          used!=u;
interface                   // Class methods

// global constructor. All ports are created free and never used
 build{i:nat}
   ^= Port{i,null,0};

// ordering of Ports
 total operator ~~(arg)
   ^=([used=arg.used]: id~~arg.id,    // if used same amount of times, ids
      []:              used~~arg.used // otherwise how many times used
     );
// output
 redefine function toString:string
   ^= ([~free]:"Port" ++ id.toString ++ " is docking " ++ (docked is Ship).toString,
       []:     "Id :" ++ id.toString ++ " is free"
      );
// get method
 function id;

// is the Port free?
 function free:bool
   ^= docked = null;
// What is docked here?
 function dockedHere:Ship||void
   ^= docked;

// Empty this Port,   (new object for Value semantics)
 function Clear:Port
   ^=Port{id,null,used+1};
// Dock at this Port, (new object for Value Semantics)
 function Dock(s:Ship):Port
   ^=Port{id,s,used};
end;
```

## A.5 Queue.pd

```
class Queue of X ^=
```

```
abstract
 var queue:seq of X;
interface
  build{}
   post queue!=seq of X{};
  build{a:seq of X}
   post queue!=a;
  function empty:bool
    ^= queue.empty;
   function head:X
    ^= queue.head;
   function tail:Queue of X
    ^= Queue of X{queue.tail};
   redefine function toString:string
    ^= queue.toString;
   function append(a:X):Queue of X
    ^= Queue of X{queue.append(a)};
end;
```

## A.6 Ship.pd

```
class Ship ^=
 abstract
  var myName:string;
  var size:ShipSize;
  var max:nat;
interface                    // Class methods
// constructor
 build{!myName:string,!size:ShipSize,!max:nat};
// get Method
 function myName;
 function size;
 function max;
 function maxSpeed:nat
  ^= max;
// output
 redefine function toString:string
   ^= "Ship - " ++ myName;
end;
```

## A.7 ShipSize.pd

```
class ShipSize ^=
  enum
    Dinghy,
    Yacht,
    Barge,
    Ferry
end;
```

## A.8 SmallShips.pd

```
class SmallShips ^= those s:Ship :- s.size <=Barge@ShipSize;
```

# Appendix B

# Library Skeleton Code

Here is the skeleton source code for the Library Database generated by the UML importer.

## B.1 Author.pd

```
class Author ^=
abstract
var name_: from string;
var myBookDescription: from BookDescription;
interface
selector name_;
function getName: from string ^= ?;
selector myBookDescription;
end;
```

## B.2 BookDescription.pd

```
class BookDescription ^=
abstract
var isbn: from nat;
var title: from string;
var myAuthor: from Author;
var mySubject: from Subject;
interface
selector isbn;
selector title;
function getIsbn: from nat ^= ?;
selector myAuthor;
selector mySubject;
end;
```

## B.3 BookInfo.pd

```
class BookInfo ^=
abstract
var myLibraryCatalog: from LibraryCatalog;
var myBorrowing: from Borrowing;
var myLibraryDB: from LibraryDB;
interface
function findBookByAuthor: from LibraryBookDescription ^= ?;
function findBookBySubject: from LibraryBookDescription ^= ?;
schema !borrowItem (copy: from LibraryItem, bor: from Borrower, date: from string) post ?;
schema !returnItem (copy: from LibraryItem) post ?;
selector myLibraryCatalog;
selector myBorrowing;
selector myLibraryDB;
end;
```

## B.4 BorrowerBase.pd

```
class BorrowerBase ^=
abstract
var myUserBase: from UserBase;
var myBorrower: from Borrower;
interface
schema !addBorrower post ?;
selector myUserBase;
selector myBorrower;
end;
```

## B.5 Borrower.pd

```
class Borrower ^= inherits Person
abstract
var borrowerId: from nat;
var limit: from nat;
var myBorrowing: from Borrowing;
var myBorrowerBase: from BorrowerBase;
interface
selector borrowerId;
selector limit;
schema !increaseLimit (inc: from nat) post ?;
selector myBorrowing;
selector myBorrowerBase;
end;
```

## B.6 Borrowing.pd

```
class Borrowing ^=
abstract
var dateBorrowed: from string;
var myBookInfo: from BookInfo;
var myLibraryItem: from LibraryItem;
var myBorrower: from Borrower;
interface
selector dateBorrowed;
selector myBookInfo;
selector myLibraryItem;
selector myBorrower;
end;
```

## B.7 LibraryBook.pd

```
class LibraryBook ^=
interface
end;
```

## B.8 LibraryBookDescription.pd

```
class LibraryBookDescription ^= inherits BookDescription
abstract
var libraryId: from nat;
var cost: from nat;
var section: from string;
var myLibraryItem: from LibraryItem;
interface
selector libraryId;
selector cost;
selector section;
function getId: from nat ^= ?;
selector myLibraryItem;
end;
```

## B.9 LibraryCatalog.pd

```
class LibraryCatalog ^=
abstract
var myBookInfo: from BookInfo;
var myLibraryItem: from LibraryItem;
interface
schema !buyBook post ?;
selector myBookInfo;
selector myLibraryItem;
end;
```

## B.10 LibraryDB.pd

```
class LibraryDB ^=
abstract
var myBookInfo: from BookInfo;
var myUserBase: from UserBase;
interface
selector myBookInfo;
selector myUserBase;
end;
```

## B.11 LibraryItem.pd

```
class LibraryItem ^=
abstract
var copyId: from nat;
var dateAcquired: from string;
var myBorrowing: from Borrowing;
var myLibraryBookDescription: from LibraryBookDescription;
var myLibraryCatalog: from LibraryCatalog;
interface
selector copyId;
```

```
selector dateAcquired;
function getId: from nat ^= ?;
selector myBorrowing;
selector myLibraryBookDescription;
selector myLibraryCatalog;
end;
```

## B.12 Person.pd

```
class Person ^=
abstract
var firstName: from string;
var lastName: from string;
var dateOfBirth: from string;
interface
selector firstName;
selector lastName;
selector dateOfBirth;
function getName: from string ^= ?;
end;
```

## B.13 Staff.pd

```
class Staff ^= inherits Person
abstract
var staffId: from nat;
var salary: from nat;
var myStaffBase: set of ref from StaffBase;
interface
selector staffId;
selector salary;
schema !increaseSalary (inc: from nat) post ?;
selector myStaffBase;
end;
```

## B.14 StaffBase.pd

```
class StaffBase ^=
abstract
var myStaff: from Staff;
var myUserBase: from UserBase;
interface
schema !addStaff post ?;
selector myStaff;
selector myUserBase;
end;
```

## B.15 Subject.pd

```
class Subject ^=
abstract
var name_: from string;
var myBookDescription: from BookDescription;
interface
selector name_;
function getName: from string ^= ?;
selector myBookDescription;
end;
```

## B.16 UserBase.pd

```
class UserBase ^=
abstract
var myBorrowerBase: from BorrowerBase;
var myStaffBase: from StaffBase;
var myLibraryDB: from LibraryDB;
interface
schema !addBorrower post ?;
schema !addStaff post ?;
schema !removeBorrower (b: from Borrower) post ?;
schema !removeStaff (s: from Staff) post ?;
selector myBorrowerBase;
selector myStaffBase;
selector myLibraryDB;
end;
```

## B.17 nat.pd

```
class nat ^=
interface
end;
```

## B.18 string.pd

```
class string ^=
interface
end;
```

# Appendix C

# Library Specification

Here is the specification of the Library Database.

## C.1 Author.pd

```
class Author ^=
abstract
    var myName:string;
interface
    build{!myName:string};

    function getName:string ^= myName;

    redefine function toString:string
        ^= getName;
end;
```

## C.2 BookDescription.pd

```
class BookDescription ^=
abstract
    var isbn:nat;
    var title:string;
    var authors:set of Author;
    var subjects:set of Subject;
interface
    build{!isbn:nat,!title:string,!authors:set of Author,!subjects:set of Subject};
    build{!isbn:nat,!title:string}
        post authors!=set of Author{},
             subjects!=set of Subject{};

    function getIsbn:nat ^= isbn;

    redefine function toString:string
```

```
        ^= "ISBN: "++isbn.toString++", "++title;

    function myName:string
        ^= toString;

    function getAuthors:set of Author
        ^= authors;

    function getSubjects:set of Subject
        ^= subjects;
end;
```

## C.3 BorrowerBase.pd

```
class BorrowerBase ^=
abstract
    var allBorrowers:set of Borrower;
    invariant forall b1,b2::allBorrowers :- b1~=b2 ==> b1.getId ~= b2.getId;
interface
    build{}
        post allBorrowers!=set of Borrower{};

    function uniqueId(b:Borrower):bool
        ^= b.getId ~in usedIds;

    function uniqueId(bId:nat):bool
        ^= bId ~in usedIds;

    function allTheBorrowers:set of Borrower
        ^= allBorrowers;

    function usedIds:set of nat
        ^= for b::allBorrowers yield b.getId;

    schema !addBorrower(bor:Borrower)
        pre self.uniqueId(bor)
        post allBorrowers!=allBorrowers.append(bor)
        assert ~self'.uniqueId(bor);
end;
```

## C.4 Borrower.pd

```
class Borrower ^=
    inherits Person
abstract
    var borrowerId:nat;
    var limit:nat;
interface
    build{first:string,last:string,dOB:string,!borrowerId:nat, !limit:nat}
        inherits Person{first,last,dOB};
    build{first:string,last:string,dOB:string,!borrowerId:nat}
        inherits Person{first,last,dOB}
        post limit!=5;

    function getId:nat
        ^= borrowerId;

    schema !increaseLimit (inc:nat)
```

```
        post limit!=limit+inc;

    redefine function toString:string
        ^= borrowerId.toString++":"++getName;
end;
```

## C.5 Borrowing.pd

```
class Borrowing ^=
abstract
    var dateBorrowed:string;
    var item: LibraryItem;
    var heldBy: Borrower;
interface
    build{!item:LibraryItem, !heldBy:Borrower,!dateBorrowed:string};

    function getItem:LibraryItem
        ^= item;

    function borrowedBy:Borrower
        ^= heldBy;

end;
```

## C.6 LibraryBookDescription.pd

```
class LibraryBookDescription ^=
    inherits BookDescription
abstract
    var libraryId:nat;
    var cost:nat;
    var section:string;
interface
    build{!libraryId:nat, !cost:nat, !section:string,
           isb:nat,titl:string,auth:set of Author,subj:set of Subject}
        inherits BookDescription{isb,titl,auth,subj};

   build{!libraryId:nat, !cost:nat, !section:string,isb:nat,titl:string}
        inherits BookDescription{isb,titl};

    function getId:nat ^= libraryId;

    redefine function toString:string
        ^= myName++","++libraryId.toString;

    function getSection:string
        ^= section;

end;
```

## C.7 LibraryCatalog.pd

```
class LibraryCatalog ^=
abstract
    var allLibraryItems: set of LibraryItem;

    function alltheBooks:set of LibraryBookDescription
```

```
        ^= for item::allLibraryItems yield item.getBook;

    invariant forall item1,item2::allLibraryItems
                                :- item1~=item2 ==> item1.getId ~= item2.getId;
    invariant #alltheBooks <= #allLibraryItems;
    invariant forall book::alltheBooks :- #(those copy::allLibraryItems
                                                :- copy.getBook=book)<=1000;

interface
    build{}
        post allLibraryItems!=set of LibraryItem{};

    function newItemId(id:nat):bool
        ^= forall items::allKnownItems :- id~=items.getId;

    function newBook(b:LibraryBookDescription):bool
        ^= b in allKnownBooks;

    function allKnownItems:set of LibraryItem
        ^= allLibraryItems;

    function allKnownBooks:set of LibraryBookDescription
        ^= alltheBooks;

    function allKnownItemIds:set of nat
        ^= for item::allKnownItems yield item.getId;

    function allKnownBookIds:set of nat
        ^= for book::allKnownBooks yield book.getId;

    function hasItem(copy:LibraryItem):bool
        ^= copy in allKnownItems;

    function uniqueItemId(id:nat):bool
        ^= id ~in allKnownItemIds;

    schema !addLibraryItem(copy:LibraryItem)
        pre ~hasItem(copy),
            uniqueItemId(copy.getId)
        post allLibraryItems!=allLibraryItems.append(copy);

    schema !removeLibraryItem(copy:LibraryItem)
        pre hasItem(copy)
        post allLibraryItems!= allLibraryItems.remove(copy);
end;
```

# C.8 LibraryDB.pd

```
class LibraryDB ^=
abstract
    var books:LibraryStock;
    var users:UserBase;
    var today:string;
interface
    build{context:Environment}
        post today!=context.getCurrentDateTime.toString,
             books!=LibraryStock{},
             users!=UserBase{};

// Functions for building information about books from a set of strings
```

```
//  ensure the string is a valid number
    function isNumber(val:string):bool
        ^= forall i::0..<#val :- val[i].isDigit & #val>0;

    function getNumber(val:string):nat
        pre isNumber(val)
        ^= nat{val};

//  get the first string before a comma from a string
    function firstComma(s:string):string
    ^= ( let n^= s.findFirst(‘,‘);
         [n<0]: s, []: s.take(n+1)
       );

//  make a set of all the authors of a book
    function makeAuthorSet(s:string):set of Author
        decrease #s
        ^= ( let stripped ^= s;
             [stripped.empty]: set of Author{},
             []:
              (let a ^= firstComma(stripped);
               makeAuthorSet(stripped.drop(#a)).append(Author{a})
              )
           );

//  as above except for subjects
    function makeSubjectSet(s:string):set of Subject
        decrease #s
        ^= ( let stripped ^= s;
             [stripped.empty]: set of Subject{},
             []:
              (let su ^= firstComma(stripped);
               makeSubjectSet(stripped.drop(#su)).append(Subject{su})
              )
           );

// function to make the new book from several strings
    function makeLibraryBook(libID,cost,section,isbn,title,authors,subjects:string):
                                                                  LibraryBookDescription
        pre isNumber(libID), isNumber(cost), isNumber(isbn)
        ^=(let authset^=makeAuthorSet(authors);
        let subjset^=makeSubjectSet(subjects);
        LibraryBookDescription{getNumber(libID),
                               getNumber(cost),
                               section,
                               getNumber(isbn),
                               title,
                               authset,
                               subjset}
          );

    function makeStaffMember(first,last,doB,id,salary:string):Staff
        pre isNumber(id),isNumber(salary)
        ^= Staff{first,last,doB,getNumber(id),getNumber(salary)};

    function makeBorrowerMember(first,last,doB,id,limit:string):Borrower
        pre isNumber(id),isNumber(limit)
        ^= Borrower{first,last,doB,getNumber(id),getNumber(limit)};

    function makeLibraryItem(id,libID,cost,section,isbn,title,
                                                authors,subjects:string):LibraryItem
```

```
    pre isNumber(id),isNumber(libID), isNumber(cost), isNumber(isbn)
    ^= (let libBook^=makeLibraryBook(libID,cost,section,isbn,title,authors,subjects);
        LibraryItem{getNumber(id),today,libBook}
       );

function makeLibraryItem(id:string,libBook:LibraryBookDescription):LibraryItem
    pre isNumber(id)
    ^= LibraryItem{getNumber(id),today,libBook};

schema !addStaff(first,last,doB,id,salary:string,rslt!:LibraryResultCode)
    post (
        [~isNumber(id)|~isNumber(salary)]: rslt!=incorrectInput@LibraryResultCode,
        []: ( let s^= makeStaffMember(first,last,doB,id,salary);
            [users.uniqueId(s)]:users!addStaff(s),
                                rslt!=success@LibraryResultCode,
            []: rslt!=knownUser@LibraryResultCode
            )
         );

schema !addBorrower(first,last,doB,id,limit:string,rslt!:LibraryResultCode)
    post (
        [~isNumber(id)|~isNumber(limit)]: rslt!=incorrectInput@LibraryResultCode,
        []: (let b^= makeBorrowerMember(first,last,doB,id,limit);
            [users.uniqueId(b)]:users!addBorrower(b),
                                rslt!=success@LibraryResultCode,
            []: rslt!=knownUser@LibraryResultCode
            )
         );

schema !nextDay(next:string)
    post today!=next;

schema !addLibraryItem(id,libID,cost,section,isbn,
                    title,authors,subjects:string, rslt!:LibraryResultCode)
    post (
        [~isNumber(id)|~isNumber(libID)|~isNumber(cost)|~isNumber(isbn)]:
                 rslt!=incorrectInput@LibraryResultCode,
        []: (let newCopy^=makeLibraryItem(id,libID,cost,section,
                                      isbn,title,authors,subjects);
            [books.hasItem(newCopy)]: rslt!=duplicateCopy@LibraryResultCode,
            []: books!addLibraryItem(newCopy),
                rslt!=success@LibraryResultCode
            )
     );

schema !removeLibraryItem(id:string, rslt!:LibraryResultCode)
    post (
        [~isNumber(id)]: rslt!=incorrectInput@LibraryResultCode,
        []: (let copyId^=getNumber(id);
            [~books.hasId(copyId)]: rslt!=notOwned@LibraryResultCode,
            []: ( let newCopy^=books.findItem(copyId);
                [~books.itemAvailable(newCopy)]: rslt!=notAvailable@LibraryResultCode,
                []:  books!removeLibraryItem(newCopy),
                     rslt!=success@LibraryResultCode
                )
            )
         );

schema !borrowLibraryItem(itemId:string, borrowerId:string, rslt!:LibraryResultCode)
    post (
        [~isNumber(itemId)|~isNumber(borrowerId)]: rslt!=incorrectInput@LibraryResultCode,
```

```
            []: (let copyId^=getNumber(itemId); let borrId^=getNumber(borrowerId);
                [~books.hasId(copyId)]: rslt!=notOwned@LibraryResultCode,
                [~users.hasId(borrId)]: rslt!=unregistered@LibraryResultCode,
                []: (let copy^=books.findItem(copyId); let borrower^=users.findUser(borrId);
                     [~books.itemAvailable(copy)]: rslt!=notAvailable@LibraryResultCode,
                     [~books.withinLimits(borrower)]: rslt!=maxLimits@LibraryResultCode,
                     []: books!borrowLibraryItem(copy,borrower,today),
                         rslt!=success@LibraryResultCode
                    )
               )
             );

    schema !returnLibraryItem(itemId:string,rslt!:LibraryResultCode)
        post (
            [~isNumber(itemId)]:rslt!=incorrectInput@LibraryResultCode,
            []: (let copyId^=getNumber(itemId);
                [~books.hasId(copyId)]: rslt!=notOwned@LibraryResultCode,
                []: (let copy^=books.findItem(copyId);
                     [books.itemAvailable(copy)]: rslt!=available@LibraryResultCode,
                     []: books!returnLibraryItem(copy),
                         rslt!=success@LibraryResultCode
                    )
               )
             );

    schema !findItemsByAuthor(author:string,booksList!:string,rslt!:LibraryResultCode)
        post ( let authset^=makeAuthorSet(author);
            [authset.empty]: booksList!="No Author",
                             rslt!=incorrectInput@LibraryResultCode,
            []: (let bookset^= books.findBookByAuthor(authset);
                    booksList!=bookset.toString,
                    rslt!=success@LibraryResultCode
               )
             );

    schema !findItemsBySubject(subject:string,booksList!:string,rslt!:LibraryResultCode)
        post ( let subjset^=makeSubjectSet(subject);
            [subjset.empty]: booksList!="No Subject",
                             rslt!=incorrectInput@LibraryResultCode,
            []: (let bookset^= books.findBookBySubject(subjset);
                    booksList!=bookset.toString,
                    rslt!=success@LibraryResultCode
               )
             );

end;
```

# C.9 LibraryItem.pd

```
class LibraryItem ^=
abstract
    var copyId:(nat in 0..999);
    var dateAcquired:string;
    var myLibraryBookDescription: LibraryBookDescription;

interface
    build{!copyId:nat,!dateAcquired:string,
          !myLibraryBookDescription:LibraryBookDescription}
        pre copyId<=999;
```

```
    function getId:nat ^= (myLibraryBookDescription.getId*1000) + copyId;

    function getBook:LibraryBookDescription
        ^= myLibraryBookDescription;


end;
```

## C.10 LibraryResultCode.pd

```
class LibraryResultCode ^=
    enum
        success,          // successful operation
        unauthorized,     // the requestor is not a member of staff
        unregistered,     // the borrower is not a member of library
        notOwned,         // the book is not owned by the library
        notAvailable,     // the book is currently checked out
        available,        // the book is available
        unknownAuthor,    // the author is not in the database
        unknownSubject,   // the subject is not in the database
        maxLimits,        // the borrower is over their borrowing limit
        duplicateCopy,    // the copy being added already exists
        duplicateBook,    // the book being added already exists
        noCopy,           // the copy isn't registered
        knownUser,        // the user exists
        neverBeenReturned,// the copy has never been returned (i.e. no previous borrower)
        incorrectInput    // the input to the wrapper class is incorrect
    end;
```

## C.11 LibraryStock.pd

```
class LibraryStock ^=
abstract
    var catalog: LibraryCatalog;
    var currentlyBorrowed: set of Borrowing;

interface
    build{}
        post catalog!=LibraryCatalog{},
             currentlyBorrowed!=set of Borrowing{};

    function checkedOut:set of LibraryItem
        ^= for b::currentlyBorrowed yield b.getItem;

    function available:set of LibraryItem
        ^= catalog.allKnownItems -- checkedOut;

    function itemAvailable(c:LibraryItem):bool
        ^= c in available;

    function booksHeld:set of LibraryBookDescription
        ^= catalog.allKnownBooks;

    function itemCurrentlyBorrowed(copy:LibraryItem):bool
        ^= copy in checkedOut;

    function copyBorrowing(copy:LibraryItem):Borrowing
        pre itemCurrentlyBorrowed(copy)
```

```
        ^= that b::currentlyBorrowed :- b.getItem = copy;

    function hasItem(copy:LibraryItem):bool
        ^= catalog.hasItem(copy);

    function hasId(id:nat):bool
        ^= id in catalog.allKnownItemIds;

    function withinLimits(bor:Borrower):bool
        ^= true;

    function findItem(id:nat):LibraryItem
        pre hasId(id)
        ^= that copy::catalog.allKnownItems :- copy.getId = id;

    function findBookByAuthor(aut:set of Author):set of LibraryBookDescription
        ^= those b::booksHeld :- aut <<= b.getAuthors;

    function findBookBySubject(sub:set of Subject):set of LibraryBookDescription
        ^= those b::booksHeld :- sub <<= b.getSubjects;

    function findCopiesBorrowedByBorrower(b:Borrower):set of LibraryItem
        ^= those item::checkedOut :- copyBorrowing(item).borrowedBy = b;

    schema !borrowLibraryItem (copy:LibraryItem, bor:Borrower, date:string)
        pre itemAvailable(copy)
        post currentlyBorrowed!=currentlyBorrowed.append(Borrowing{copy,bor,date});

    schema !returnLibraryItem (copy:LibraryItem)
        pre itemCurrentlyBorrowed(copy)
        post currentlyBorrowed!=currentlyBorrowed.remove(copyBorrowing(copy));

    schema !addLibraryItem(copy:LibraryItem)
        pre ~hasItem(copy)
        post catalog!addLibraryItem(copy);

    schema !removeLibraryItem(copy:LibraryItem)
        pre hasItem(copy),
            itemAvailable(copy)
        post catalog!removeLibraryItem(copy);

end;
```

## C.12 Person.pd

```
class Person ^=
abstract
    var firstName:string;
    var lastName:string;
    var dateOfBirth:string;
interface
    build{!firstName:string,!lastName:string,!dateOfBirth:string};

    function getName:string
        ^= firstName++" "++lastName;

    redefine function toString:string
        ^= getName;
end;
```

## C.13 Staff.pd

```
class Staff ^=
    inherits Person
abstract
    var staffId:nat;
    var salary:nat;
interface
    build{first:string,last:string,dOB:string,!staffId:nat,!salary:nat}
        inherits Person{first,last,dOB};

    redefine function toString:string
        ^= staffId.toString++":"++getName;

    function getId:nat
        ^= staffId;

    schema !increaseSalary (inc:nat)
        post salary!=salary+inc;


end;
```

## C.14 StaffBase.pd

```
class StaffBase ^=
abstract
    var allStaff:set of Staff;

    invariant forall s1,s2::allStaff :- s1~=s2 ==> s1.getId ~= s2.getId
interface
    build{}
        post allStaff!=set of Staff{};

    function uniqueId(s:Staff):bool
        ^= s.getId ~in usedIds;

    function usedIds:set of nat
        ^= for s::allStaff yield s.getId;

    schema !addStaff(sta:Staff)
        pre uniqueId(sta)
        post allStaff!=allStaff.append(sta);
end;
```

## C.15 Subject.pd

```
class Subject ^=
abstract
    var myName:string;
interface
    build{!myName:string};

    function getName:string
        ^= myName;

    redefine function toString:string
        ^= getName;
end;
```

## C.16 UserBase.pd

```
class UserBase ^=
abstract
    var allBorrowers: BorrowerBase;
    var allStaff: StaffBase;

interface

    build{}
        post allBorrowers!=BorrowerBase{},
             allStaff!=StaffBase{};

    function usedIds:set of nat
        ^= allBorrowers.usedIds ++ allStaff.usedIds;

    function uniqueId(u:Borrower):bool
        ^= ~hasId(u.getId);

    function uniqueId(u:Staff):bool
        ^= ~hasId(u.getId);

    function hasId(uId:nat):bool
        ^= uId in usedIds;

    function findUser(id:nat):Borrower
        pre hasId(id)
        ^= that user::allBorrowers.allTheBorrowers :- user.getId = id;

    schema !addBorrower(bor:Borrower)
        pre uniqueId(bor)
        post allBorrowers!addBorrower(bor);

    schema !addStaff(sta:Staff)
        pre uniqueId(sta)
        post allStaff!addStaff(sta);

end;
```

# Appendix D

# Library Refinement

Here is the refined Perfect source code files of the Library Database Software.

## D.1 HashedBucket.pd

```
class HashedBucket of X require X has operator =(arg);
                                    function hash:nat  end^=
abstract
    var
        hb:map of (nat ->set of X),
        // the table is represented as a map of natural numbers to sets
        // this prevents the creation of empty reduntant entries
        // necessary as perfect has dynamic allocation of sequences
        // (Not arrays)
        hashSize:nat;       // keep track of the max size of the hash table

        invariant hashSize>0;
        // all the keys are in the range of 0 to size
        invariant forall x::hb.dom :- 0<=x<=hashSize;
        // the hashtable is never bigger than the size
        invariant #hb <= hashSize;

        // all the elements in the hb have valid keys
        // (says nothing about elements not in hb)
        // needed? to prove being a member => the key is in the hb for removal
        axiom assert forall a:X :- a in self ==> hash(a) in hb.dom;

confined
    // methods accessible to the object only

    // add an element that has a new hash key
    schema !addNewHash(pos:nat,a:X)
        pre 0<=pos<=hashSize,       // key in range
            pos ~in hb.dom,         // key not in table
            #hb < hashSize          // we have space in the hashtable
        post (let newMap^=set of X{a};      // new element
```

```
            hb!=hb.append(pair of (nat,set of X){pos,newMap})
           )
        assert forall elem::self.ran :- elem in self',
               a in self';

    // add an element whose key is in the system
    schema !addOldHash(pos:nat,a:X)
        pre 0<=pos<=hashSize,       // key in range
            pos in hb.dom           // key in table
        post (let newMap^=hb[pos].append(a);    // old element + new element
              hb!=hb.remove(pos) then           // remove the old
              hb!=hb.append(pair of (nat,set of X){pos,newMap})
                                                // add everything back
           )
        assert forall elem::self.ran :- elem in self',
               a in self';

interface

    // construct an empty hashedbucket
    build{!hashSize:nat}
        pre hashSize>0
        post hb!=map of (nat ->set of X){};

//**Some standard operations for a data structure**//

    // get all the elements of the hashedbucket
    function ran:set of X
        ^= flatten(hb.ran);

    // get all the keys used in the hashedbucket
    function keysUsed:set of nat
        ^= hb.dom;

    // is the hashedbucket empty
    function empty:bool
        ^= hb.empty;

    // how many elements are in the hashedbucket
    operator #:nat
        ^= #ran;

    // what is the maximum Hash value
    function maxHashSize:nat
        ^= hashSize;

    // is the element in the hashtable
    operator (a:X)in:bool
        ^= ([hash(a) in  hb.dom]:       // is the key valid?
                a in hb[hash(a)],       // is the element in that bucket
            []:
                false
           );

    // undefined hash function. Ensures the result is in the range though
    // We might want the classes that use the system to define their own hash
    // function. Problems will occur in 1) building the code; 2) Primitive types
    function hash(a:X):nat
        satisfy 0<=result<=hashSize
        via
            value (a.hash)% hashSize
```

```
        end;

    // add an element to the hashedbucket
    schema !add(a:X)
        post (let pos^= hash(a);
                [pos in hb.dom]:            // we have that key
                    !addOldHash(pos,a),
                []: //#hb < hashSize]:      // we have space for new hash
                    !addNewHash(pos,a)
             )
        assert forall elem::self.ran :- elem in self',
               a in self';

    // remove an element from the hashedbucket provided it is already there
    schema !remove(a:X)
        pre a in self
        post (let pos^= hash(a);
              let newMap^=hb[pos].remove(a);        // new bucket, without element
              hb!=hb.remove(pos) then               // remove old bucket
              hb!=hb.append(pair of(nat,set of X){pos,newMap})   // all new bucket
             )
        assert forall elem::self'.ran :- elem in self,
               a ~in self';

end;
```

# D.2 PriorityQueue.pd

```
class PriorityQueue of X require X has operator =(arg);
                                      function priority:nat end ^=
abstract
    var
        myQueue:Heap of X;
interface
    build{}
        post myQueue!=Heap of X{};

    schema !insert(a:X)
        post myQueue!insert(a);

    function getElement:X
        pre ~myQueue.empty
        ^= myQueue.largest;

    schema !remove
        pre  ~myQueue.empty
        post myQueue!remove(getElement);

    function ran:set of X
        ^= myQueue.ran;
end;

//******************************************************
//* File:    C:\Projects\Perfect\Heap\Heap.pd
//* Author:  Gareth Carter
//* Created: 12:34:53 on Monday February 23rd 2004 UTC
//******************************************************

class Heap of X require X has  operator =(arg);
                               function priority:nat end ^=
```

```
abstract
    var
        l:seq of X;

    invariant isHeap(l);

confined

    nonmember function heapify(xs:seq of X,low,high:nat):seq of X
        pre ~xs.empty,
            0<low<=high<=#xs
        decrease high-low
        ^= (let large^=2*low; assert large>1;
             [large<=high]:
                ([large<high
                & xs[<large].priority<xs[large].priority
                & xs[<low].priority < xs[large].priority]:
                    heapify(exchange(xs,low,large+1),large+1,high),
                 [xs[<low].priority < xs [<large].priority]:
                    heapify(exchange(xs,low,large),large,high),
                 []:
                    xs
                ),
             []:
                xs
           );

    function sift_up(xs:seq of X,pos:nat):seq of X
        pre 0<pos<=#xs
        decrease pos
        ^= (let parentPos^= pos/2;
            [pos = 1]:          // at root
                xs,
            [xs[<pos].priority<=xs[<parentPos].priority]:
                 // my priority less than parents
                xs,
            []:         // my priority greater than parents
                sift_up(exchange(xs,parentPos,pos),parentPos)
           )
           ;

    function sift_down(xs:seq of X):seq of X
        pre ~xs.empty
        ^= ([#xs=1]:
                seq of X{},
            []:
                heapify(exchange(xs,1,#xs).front,1,<#xs)
           );

interface
    build{}
        post l!=seq of X{};
// Start of HEAP DEFINITION //
    nonmember function isHeap(a:seq of X):bool
        ^= forall p::1..(#a/2) :- greaterPriority(p,a);

    nonmember function greaterPriority(p:nat,a:seq of X):bool
        pre 0<p<=#a
        ^= (let child^=2*p;
            [child<#a]:         // has two children
                a[<p].priority>=a[<child].priority
```

```
                & a[<p].priority>=a[<child+1].priority,
            [child=#a]:         // has one child
                a[<p].priority>=a[<child].priority,
            []:                 // has no children
                true
          );
// End of HEAP DEFINITION //

    nonmember function exchange(xs:seq of X,i:nat,j:nat):seq of X
        pre 0<i<j<=#xs
        ^= xs.take(<i).append(xs[<j])++
           xs.take(<j).drop(i).append(xs[<i])++
           xs.take(#xs).drop(j)
        assert result.ranb = xs.ranb;

    nonmember schema swap(xs!:seq of X,i:nat,j:nat)
        pre 0<i<j<=#xs
        post (let temp^= xs[<i];
              xs[<i]!=xs[<j] then
              xs[<j]!=temp
             )
        assert xs.ranb = xs'.ranb;

    schema !insert(a:X)
        post l!=sift_up(l.append(a),#l+1)
        assert forall elem::self.ran :- elem in self',
               a in self';

    schema !remove(a:X)
        pre ~empty,
            largest=a
        post l!=sift_down(l)
        assert forall elem::self'.ran :- elem in self,
               a ~in self';

    function largest:X
        pre ~empty
        ^= l.head;

    function empty:bool
        ^= l.empty;

    redefine function toString:string
        ^= l.toString;

    function ran:set of X
        ^= l.ran;

    operator (a:X)in:bool
        ^= a in  l;

end;
\begin{verbatim}
```

## D.3 LibraryCatalog.pd

```
class LibraryCatalog ^=
abstract
    var allLibraryItems: set of LibraryItem;
```

```
    ghost function alltheBooks:set of LibraryBookDescription
        ^= for item::allLibraryItems yield item.getBook;

    invariant forall item1,item2::allLibraryItems
                                    :- item1~=item2 ==> item1.getId ~= item2.getId;
    invariant #alltheBooks <= #allLibraryItems;
     invariant forall book::alltheBooks :- #(those copy::allLibraryItems
                                                    :- copy.getBook=book)<=1000;

internal

    var
        allLibItems : HashedBucket of LibraryItem;

    function allLibraryItems
        ^= allLibItems.ran;

interface
    operator =(arg);

    build{}
        post allLibraryItems!=set of LibraryItem{}
        via allLibItems!=HashedBucket of LibraryItem{1000}
        end;

    function newItemId(id:nat):bool
        ^= forall items::allKnownItems :- id~=items.getId;

    function newBook(b:LibraryBookDescription):bool
        ^= b in allKnownBooks;

    function allKnownItems:set of LibraryItem
        ^= allLibraryItems
        via
            value allLibItems.ran
        end;

    function allKnownBooks:set of LibraryBookDescription
        ^= alltheBooks
        via
            value for item::allKnownItems yield item.getBook;
        end;

    function allKnownItemIds:set of nat
        ^= for item::allKnownItems yield item.getId;

    function allKnownBookIds:set of nat
        ^= for book::allKnownBooks yield book.getId;

    function hasItem(copy:LibraryItem):bool
        ^= copy in allKnownItems
           via
                value (copy in allLibItems)
           end;

    function uniqueItemId(id:nat):bool
        ^= id ~in allKnownItemIds;

    schema !addLibraryItem(copy:LibraryItem)
        pre ~hasItem(copy),
            uniqueItemId(copy.getId)
```

```
        post allLibraryItems!=allLibraryItems.append(copy)
        via
            allLibItems!add(copy)
        end;

    schema !removeLibraryItem(copy:LibraryItem)
        pre hasItem(copy)
        post allLibraryItems!= allLibraryItems.remove(copy)
        via allLibItems!remove(copy)
        end;
end;
```

## D.4 LibraryItem.pd

```
class LibraryItem ^=
abstract
    var copyId:(nat in 0..999);
    var dateAcquired:string;
    var myLibraryBookDescription: LibraryBookDescription;

interface
   build{!copyId:nat,!dateAcquired:string,!myLibraryBookDescription:LibraryBookDescription}
        pre copyId<=999;

    function getId:nat ^= (myLibraryBookDescription.getId*1000) + copyId;

    function getBook:LibraryBookDescription
        ^= myLibraryBookDescription;

    function hash:nat
       ^= getId;

end;
```

## D.5 LibraryStock.pd

```
class LibraryStock ^=
abstract
    var catalog: LibraryCatalog;
    var currentlyBorrowed: set of Borrowing;

internal
    var
        borrowed:PriorityQueue of Borrowing;

    function currentlyBorrowed
        ^= borrowed.ran;

interface
    operator =(arg);

    build{}
        post catalog!=LibraryCatalog{},
             currentlyBorrowed!=set of Borrowing{}
        via  catalog!=LibraryCatalog{},
             borrowed!=PriorityQueue of Borrowing{}
        end;
```

```
function checkedOut:set of LibraryItem
    ^= for b::currentlyBorrowed yield b.getItem
    via value(for b::borrowed.ran yield b.getItem)
    end;

function available:set of LibraryItem
    ^= catalog.allKnownItems -- checkedOut;

function itemAvailable(c:LibraryItem):bool
    ^= c in available;

function booksHeld:set of LibraryBookDescription
    ^= catalog.allKnownBooks;

function itemCurrentlyBorrowed(copy:LibraryItem):bool
    ^= copy in checkedOut;

function copyBorrowing(copy:LibraryItem):Borrowing
    pre itemCurrentlyBorrowed(copy)
    ^= that b::currentlyBorrowed :- b.getItem = copy
    via value(that b::borrowed.ran :- b.getItem = copy)
    end;

function hasItem(copy:LibraryItem):bool
    ^= catalog.hasItem(copy);

function hasId(id:nat):bool
    ^= id in catalog.allKnownItemIds;

function withinLimits(bor:Borrower):bool
    ^= true;

function findItem(id:nat):LibraryItem
    pre hasId(id)
    ^= that copy::catalog.allKnownItems :- copy.getId = id;

function findBookByAuthor(aut:set of Author):set of LibraryBookDescription
    ^= those b::booksHeld :- aut <<= b.getAuthors;

function findBookBySubject(sub:set of Subject):set of LibraryBookDescription
    ^= those b::booksHeld :- sub <<= b.getSubjects;

function findCopiesBorrowedByBorrower(b:Borrower):set of LibraryItem
    ^= those item::checkedOut :- copyBorrowing(item).borrowedBy = b;

schema !borrowLibraryItem (copy:LibraryItem, bor:Borrower, date:string)
    pre itemAvailable(copy)
    post currentlyBorrowed!=currentlyBorrowed.append(Borrowing{copy,bor,date})
    via  borrowed!insert(Borrowing{copy,bor,date})
    end;

schema !returnLibraryItem (copy:LibraryItem)
    pre itemCurrentlyBorrowed(copy)
    post currentlyBorrowed!=currentlyBorrowed.remove(copyBorrowing(copy))
    via  borrowed!remove
    end;

schema !addLibraryItem(copy:LibraryItem)
    pre ~hasItem(copy)
    post catalog!addLibraryItem(copy);
```

```
    schema !removeLibraryItem(copy:LibraryItem)
        pre hasItem(copy),
            itemAvailable(copy)
        post catalog!removeLibraryItem(copy);

end;
```

# Appendix E

# Library Java Implementation

Here is the Java source code of the Library Database.

## E.1 Author.java

```
package oolibrary;


import Ertsys.*;

// Packages imported
import oolibrary.*;


public class Author extends _eAny
{
    protected _eSeq myName;
    public Author (_eSeq _vmyName, char _t0_vmyName)
    {
        super ();
        myName = _vmyName;
    }

    public _eSeq getName ()
    {
        return myName;
    }

    public _eSeq _rtoString ()
    {
        return getName ();
    }

    public boolean _lEqual (Author _vArg_11_9)
    {
        if (this == _vArg_11_9) return true;
```

```
        return _vArg_11_9.myName._lEqual (myName);
    }

    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () == Author.class &&
            _lEqual ((Author) _lArg));
    }

}


// End of file.
```

## E.2 BookDescription.java

```
package oolibrary;


import Ertsys.*;

// Packages imported
import oolibrary.*;


public class BookDescription extends _eAny
{
    protected int isbn;
    protected _eSeq title;
    protected _eSet authors;
    protected _eSet subjects;
    public BookDescription (int _visbn, _eSeq _vtitle, char _t0_vtitle, _eSet _vauthors,
        Author _t0_vauthors, _eSet _vsubjects, Subject _t0_vsubjects)
    {
        super ();
        isbn = _visbn;
        title = _vtitle;
        authors = _vauthors;
        subjects = _vsubjects;
    }

    public BookDescription (int _visbn, _eSeq _vtitle, char _t0_vtitle)
    {
        super ();
        isbn = _visbn;
        title = _vtitle;
        authors = new _eSet ();
        subjects = new _eSet ();
    }

    public int getIsbn ()
    {
        return isbn;
    }

    public _eSeq _rtoString ()
    {
        return _eSystem._lString ("ISBN: ")._oPlusPlus (_eSystem._ltoString (isbn), (
            _eTemplate_0) null)._oPlusPlus (_eSystem._lString (", "), (_eTemplate_0) null)
            ._oPlusPlus (title, (_eTemplate_0) null);
```

```
    }

    public _eSeq myName ()
    {
        return _rtoString ();
    }

    public _eSet getAuthors ()
    {
        return authors;
    }

    public _eSet getSubjects ()
    {
        return subjects;
    }

    public boolean _lEqual (BookDescription _vArg_11_9)
    {
        if (this == _vArg_11_9) return true;
        return (((_vArg_11_9.authors._lEqual (authors) && _vArg_11_9.subjects._lEqual (
            subjects)) && _vArg_11_9.title._lEqual (title)) && (_vArg_11_9.isbn == isbn));
    }

    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () == BookDescription.
            class && _lEqual ((BookDescription) _lArg));
    }

}

// End of file.
```

## E.3 BorrowerBase.java

```
package oolibrary;


import Ertsys.*;

// Packages imported
import oolibrary.*;


public class BorrowerBase extends _eAny
{
    final void _lc_BorrowerBase (String _lArg)
    {
        if (_eSystem.enableClassInvariantItem && _eSystem.currentCheckNesting <= _eSystem.
            maxCheckNesting)
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                boolean _vQuantifierResult_12_15;
                {
                    _vQuantifierResult_12_15 = true;
                    int _vCaptureCount_b1_12_29 = allBorrowers._oHash ();
                    int _vLoopCounter_12_22 = 0;
```

```
                for (;;)
                {
                    if (((_vLoopCounter_12_22 == _vCaptureCount_b1_12_29) || (!
                        _vQuantifierResult_12_15))) break;
                    boolean _vQuantifierResult_12_25;
                    {
                        _vQuantifierResult_12_25 = true;
                        int _vCaptureCount_b2_12_29 = allBorrowers._oHash ();
                        int _vLoopCounter_12_25 = 0;
                        for (;;)
                        {
                            if (((_vLoopCounter_12_25 == _vCaptureCount_b2_12_29) || (
                                !_vQuantifierResult_12_25))) break;
                            _vQuantifierResult_12_25 = (!(!((Borrower) allBorrowers.
                                _oIndex (_vLoopCounter_12_22))._lEqual (((Borrower)
                                allBorrowers._oIndex (_vLoopCounter_12_25)))) || (!(((
                                Borrower) allBorrowers._oIndex (_vLoopCounter_12_22)).
                                getId () == ((Borrower) allBorrowers._oIndex (
                                _vLoopCounter_12_25)).getId ())));
                            if ((!_vQuantifierResult_12_25))
                            {
                            }
                            else
                            {
                                _vLoopCounter_12_25 = _eSystem._oSucc (
                                    _vLoopCounter_12_25);
                            }
                        }
                    }
                    _vQuantifierResult_12_15 = _vQuantifierResult_12_25;
                    if ((!_vQuantifierResult_12_15))
                    {
                    }
                    else
                    {
                        _vLoopCounter_12_22 = _eSystem._oSucc (_vLoopCounter_12_22);
                    }
                }
            }
            if (!(_vQuantifierResult_12_15)) throw new _xClassInvariantItem (
                "BorrowerBase.pd:12,15", _lArg);
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
}

void _lClassInvariantCheck (String _lArg)
{
    _lc_BorrowerBase (_lArg);
}

protected _eSet allBorrowers;
public BorrowerBase ()
{
    super ();
    allBorrowers = new _eSet ();
    _lc_BorrowerBase ("BorrowerBase.pd:15,14");
}
```

```
public boolean uniqueId (Borrower b)
{
    return (!usedIds ()._ovIn (((_eAny) new _eWrapper_int (b.getId ()))));
}

public boolean uniqueId (int bId)
{
    return (!usedIds ()._ovIn (((_eAny) new _eWrapper_int (bId))));
}

public _eSet allTheBorrowers ()
{
    return allBorrowers;
}

public _eSet usedIds ()
{
    _eSet _vForYield_27_12;
    {
        _vForYield_27_12 = new _eSet ();
        int _vCaptureCount_b_27_19 = allBorrowers._oHash ();
        int _vLoopCounter_27_16 = 0;
        for (;;)
        {
            if ((_vLoopCounter_27_16 == _vCaptureCount_b_27_19)) break;
            _vForYield_27_12 = _vForYield_27_12.append (((_eAny) new _eWrapper_int (((
                Borrower) allBorrowers._oIndex (_vLoopCounter_27_16)).getId ())));
            _vLoopCounter_27_16 = _eSystem._oSucc (_vLoopCounter_27_16);
        }
    }
    return _vForYield_27_12;
}

public void addBorrower (Borrower bor)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(uniqueId (bor))) throw new _xPre ("BorrowerBase.pd:30,18");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    allBorrowers = allBorrowers.append (((_eAny) bor));
    _lClassInvariantCheck ("BorrowerBase.pd:31,14");
}

public boolean _lEqual (BorrowerBase _vArg_11_9)
{
    if (this == _vArg_11_9) return true;
    return _vArg_11_9.allBorrowers._lEqual (allBorrowers);
}

public boolean equals (_eAny _lArg)
{
```

```
        return _lArg == this || (_lArg != null && _lArg.getClass () == BorrowerBase.class
            && _lEqual ((BorrowerBase) _lArg));
    }

}

// End of file.
```

## E.4 Borrower.java

```
package oolibrary;


import Ertsys.*;

// Packages imported
import oolibrary.*;


public class Borrower extends Person
{
    protected int borrowerId;
    protected int limit;
    public Borrower (_eSeq first, char _t0first, _eSeq last, char _t0last, _eSeq dOB, char
        _t0dOB, int _vborrowerId, int _vlimit)
    {
        super (first, (char) 0, last, (char) 0, dOB, (char) 0);
        borrowerId = _vborrowerId;
        limit = _vlimit;
    }

    public Borrower (_eSeq first, char _t0first, _eSeq last, char _t0last, _eSeq dOB, char
        _t0dOB, int _vborrowerId)
    {
        super (first, (char) 0, last, (char) 0, dOB, (char) 0);
        borrowerId = _vborrowerId;
        limit = 5;
    }

    public int getId ()
    {
        return borrowerId;
    }

    public void increaseLimit (int inc)
    {
        limit = (limit + inc);
    }

    public _eSeq _rtoString ()
    {
        return _eSystem._ltoString (borrowerId)._oPlusPlus (_eSystem._lString (":"), (
            _eTemplate_0) null)._oPlusPlus (getName (), (_eTemplate_0) null);
    }

    public boolean _lEqual (Borrower _vArg_12_9)
    {
        if (this == _vArg_12_9) return true;
        if (!super._lEqual (_vArg_12_9)) return false;
        return ((_vArg_12_9.borrowerId == borrowerId) && (_vArg_12_9.limit == limit));
```

```
    }

    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () == Borrower.class &&
            _lEqual ((Borrower) _lArg));
    }

}


// End of file.
```

## E.5 Borrowing.java

```
package oolibrary;


import Ertsys.*;

// Packages imported
import oolibrary.*;


public class Borrowing extends _eAny
{
    protected _eSeq dateBorrowed;
    protected LibraryItem item;
    protected Borrower heldBy;
    public Borrowing (LibraryItem _vitem, Borrower _vheldBy, _eSeq _vdateBorrowed, char
        _t0_vdateBorrowed)
    {
        super ();
        item = _vitem;
        heldBy = _vheldBy;
        dateBorrowed = _vdateBorrowed;
    }

    public LibraryItem getItem ()
    {
        return item;
    }

    public Borrower borrowedBy ()
    {
        return heldBy;
    }

    public int priority ()
    {
        return 1;
    }

    public boolean _lEqual (Borrowing _vArg_11_9)
    {
        if (this == _vArg_11_9) return true;
        return ((_vArg_11_9.item._lEqual (item) && _vArg_11_9.heldBy._lEqual (heldBy)) &&
            _vArg_11_9.dateBorrowed._lEqual (dateBorrowed));
    }
```

```
    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () == Borrowing.class &&
            _lEqual ((Borrowing) _lArg));
    }

}


// End of file.
```

## E.6 HashedBucket.java

```
package oolibrary;


import Ertsys.*;

// Packages imported
import oolibrary.*;


class _n1_HashedBucket extends _eAny
{
    final void _lc_HashedBucket (String _lArg)
    {
        if (_eSystem.enableClassInvariantItem && _eSystem.currentCheckNesting <= _eSystem.
            maxCheckNesting)
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                if (!((0 < hashSize))) throw new _xClassInvariantItem (
                    "HashedBucket.pd:23,27", _lArg);
            }
            catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
        if (_eSystem.enableClassInvariantItem && _eSystem.currentCheckNesting <= _eSystem.
            maxCheckNesting)
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                boolean _vQuantifierResult_25_19;
                {
                    _eSet _vCaptureBound_x_25_32 = hb.dom ();
                    _vQuantifierResult_25_19 = true;
                    int _vCaptureCount_x_25_32 = _vCaptureBound_x_25_32._oHash ();
                    int _vLoopCounter_25_26 = 0;
                    for (;;)
                    {
                        if (((_vLoopCounter_25_26 == _vCaptureCount_x_25_32) || (!
                            _vQuantifierResult_25_19))) break;
                        _vQuantifierResult_25_19 = ((((_eWrapper_int)
                            _vCaptureBound_x_25_32._oIndex (_vLoopCounter_25_26)).value <=
                            hashSize) && (0 <= ((_eWrapper_int) _vCaptureBound_x_25_32.
                            _oIndex (_vLoopCounter_25_26)).value));
```

```
                    if ((!_vQuantifierResult_25_19))
                    {
                    }
                    else
                    {
                        _vLoopCounter_25_26 = _eSystem._oSucc (_vLoopCounter_25_26);
                    }
                }
            }
            if (!(_vQuantifierResult_25_19)) throw new _xClassInvariantItem (
                "HashedBucket.pd:25,19", _lArg);
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    if (_eSystem.enableClassInvariantItem && _eSystem.currentCheckNesting <= _eSystem.
        maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!((hb._oHash () <= hashSize))) throw new _xClassInvariantItem (
                "HashedBucket.pd:27,23", _lArg);
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
}

void _lClassInvariantCheck (String _lArg)
{
    _lc_HashedBucket (_lArg);
}

protected _eMap hb;
protected int hashSize;
protected void addNewHash (int pos, _eAny a)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(((0 <= pos) && (pos <= hashSize)))) throw new _xPre (
                "HashedBucket.pd:39,14");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
```

```
        {
            if (!((!hb.dom ()._ovIn (((_eAny) new _eWrapper_int (pos)))))) throw new
                _xPre ("HashedBucket.pd:40,17");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!((hb._oHash () < hashSize))) throw new _xPre ("HashedBucket.pd:41,17"
                );
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    _eSet _vLet_newMap_42_19 = new _eSet (a);
    hb = hb.append (new _ePair (((_eAny) new _eWrapper_int (pos)), ((_eAny)
        _vLet_newMap_42_19)), (_eTemplate_0) null, (_eTemplate_1) null);
    _lClassInvariantCheck ("HashedBucket.pd:42,14");
}

protected void addOldHash (int pos, _eAny a)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(((0 <= pos) && (pos <= hashSize)))) throw new _xPre (
                "HashedBucket.pd:50,14");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(hb.dom ()._ovIn (((_eAny) new _eWrapper_int (pos))))) throw new
                _xPre ("HashedBucket.pd:51,17");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    _eSet _vLet_newMap_52_19 = ((_eSet) hb._oIndex (((_eAny) new _eWrapper_int (pos)))
        ).append (a);
```

```
    hb = hb.remove (((_eAny) new _eWrapper_int (pos)));
    hb = hb.append (new _ePair (((_eAny) new _eWrapper_int (pos)), ((_eAny)
        _vLet_newMap_52_19)), (_eTemplate_0) null, (_eTemplate_1) null);
    _lClassInvariantCheck ("HashedBucket.pd:52,14");
}

public _n1_HashedBucket (int _vhashSize)
{
    super ();
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!((0 < _vhashSize))) throw new _xPre ("HashedBucket.pd:64,21");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    hashSize = _vhashSize;
    hb = new _eMap ();
    _lc_HashedBucket ("HashedBucket.pd:65,14");
}

public _eSet ran ()
{
    return Ertsys.RtsGlobals.flatten (hb.ran (), (_eSet) null, (_eTemplate_0) null);
}

public _eSet keysUsed ()
{
    return hb.dom ();
}

public boolean empty ()
{
    return hb.empty ();
}

public int _oHash ()
{
    return ran ()._oHash ();
}

public int maxHashSize ()
{
    return hashSize;
}

public boolean _ovIn (_eAny a)
{
    return (hb.dom ()._ovIn (((_eAny) new _eWrapper_int (hash (a)))) ?
    ((_eSet) hb._oIndex (((_eAny) new _eWrapper_int (hash (a))))._ovIn (a) :
    false);
}

public int hash (_eAny a)
{
```

```
        return _eSystem._oMod (a.hash (), hashSize);
    }

    public void add (_eAny a)
    {
        int _vLet_pos_108_19 = hash (a);
        if (hb.dom ()._ovIn (((_eAny) new _eWrapper_int (_vLet_pos_108_19))))
        {
            addOldHash (_vLet_pos_108_19, a);
        }
        else
        {
            addNewHash (_vLet_pos_108_19, a);
        }
        _lClassInvariantCheck ("HashedBucket.pd:108,14");
    }

    public void remove (_eAny a)
    {
        if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
            )
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                if (!(this._ovIn (a))) throw new _xPre ("HashedBucket.pd:119,15");
            }
            catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
        int _vLet_pos_120_19 = hash (a);
        _eSet _vLet_newMap_121_19 = ((_eSet) hb._oIndex (((_eAny) new _eWrapper_int (
            _vLet_pos_120_19)))).remove (a);
        hb = hb.remove (((_eAny) new _eWrapper_int (_vLet_pos_120_19)));
        hb = hb.append (new _ePair (((_eAny) new _eWrapper_int (_vLet_pos_120_19)), ((
            _eAny) _vLet_newMap_121_19)), (_eTemplate_0) null, (_eTemplate_1) null);
        _lClassInvariantCheck ("HashedBucket.pd:120,14");
    }

    public boolean _lEqual (_n1_HashedBucket _vArg_16_9)
    {
        if (this == _vArg_16_9) return true;
        return (_vArg_16_9.hb._lEqual (hb) && (_vArg_16_9.hashSize == hashSize));
    }

    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () == _n1_HashedBucket.
            class && _lEqual ((_n1_HashedBucket) _lArg));
    }

}


// End of file.
```

## E.7 LibraryAccess.java

```
// Simple Java Swing application to front-end a Perfect Developer program

package oolibrary;

import java.io.IOException;
import java.io.File;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import Ertsys.*;

// This is the top-level class
public class LibraryAccess implements ActionListener
{
    JFrame userFrame,catalogFrame,stockFrame;
    JPanel userPanel,catalogPanel,stockPanel;

    // Staff Database Frame
    // Person information - Borrowers and Staff
    JTextField firstName,lastName,dateOfBirth,userId,salary,limit;
    JButton addStaff,addBorrower;

    // LibraryCatalog Frame
    // Book Decription Information
    JTextField libId,cost,section, isbn, bookTitle,author,subject;

    // Library Item Information
    JTextField copyId;
    JButton addCopy,removeCopy;

    // LibraryStock Frame
    // id input parameters
    JTextField staffIdQ, borrowerIdQ, copyIdQ, bookIdQ, authorQ, subjectQ;
    JButton borrowCopy,returnCopy;

    JButton findAuthor,findSubject;
    JButton whatBorrowedBy,whoBorrowed;

    // output for all windows
    JLabel uStatus,cStatus,sStatus;

    LibraryDB backend;

    // Constructor
    public LibraryAccess(Environment context)
        {
            // Create the back end
            backend = new LibraryDB(context);

            // Create the frame and container.
            userFrame = new JFrame("User Database Control");
            userPanel = new JPanel();
            userPanel.setLayout(new GridLayout(10,2));
            catalogFrame = new JFrame("Catalog Control");
            catalogPanel = new JPanel();
            catalogPanel.setLayout(new GridLayout(10,2));
            stockFrame = new JFrame("Stock Control");
            stockPanel = new JPanel();
```

```
        stockPanel.setLayout(new GridLayout(10,2));

        // Add the widgets.
        addWidgets();

        // Add the panel to the frame.
        userFrame.getContentPane().add(userPanel);
        catalogFrame.getContentPane().add(catalogPanel);
        stockFrame.getContentPane().add(stockPanel);

        // Exit when the window is closed.
        userFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        catalogFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        stockFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Show the app.
        userFrame.pack();
        userFrame.setVisible(true);
        catalogFrame.pack();
        catalogFrame.setVisible(true);
        stockFrame.pack();
        stockFrame.setVisible(true);
}

// Create and add the widgets for app.
private void addWidgets()
    {
        // Create widgets.
        firstName = new JTextField(20);
        lastName = new JTextField(20);
        dateOfBirth = new JTextField(20);
        userId = new JTextField(20);
        salary = new JTextField(20);
        limit = new JTextField(20);
        addStaff = new JButton("Add Staff");
        addBorrower = new JButton("Add Borrower");

        libId = new JTextField(20);
        cost = new JTextField(20);
        section = new JTextField(20);
        isbn = new JTextField(20);
        bookTitle = new JTextField(20);
        author = new JTextField(20);
        subject = new JTextField(20);
        copyId = new JTextField(20);
        addCopy = new JButton("Add Copy");
        removeCopy = new JButton("Remove Copy");

        staffIdQ = new JTextField(20);
        borrowerIdQ = new JTextField(20);
        copyIdQ = new JTextField(20);
        bookIdQ = new JTextField(20);
        authorQ = new JTextField(20);
        subjectQ = new JTextField(20);
        borrowCopy = new JButton("Borrow Copy");
        returnCopy = new JButton("Return Copy");
        findAuthor = new JButton("Find books by Author");
        findSubject = new JButton("Find books by Subject");
        whatBorrowedBy = new JButton("Find copies borrowed by a borrower");
        whoBorrowed = new JButton("Find who borrowed a copy last");
```

```
uStatus = new JLabel("Result", SwingConstants.LEFT);
cStatus = new JLabel("Result", SwingConstants.LEFT);
sStatus = new JLabel("Result", SwingConstants.LEFT);

// Listen to events from Convert button.
addStaff.addActionListener(this);
addBorrower.addActionListener(this);
addCopy.addActionListener(this);
removeCopy.addActionListener(this);
borrowCopy.addActionListener(this);
returnCopy.addActionListener(this);
findAuthor.addActionListener(this);
findSubject.addActionListener(this);
whatBorrowedBy.addActionListener(this);
whoBorrowed.addActionListener(this);

// Add widgets to container.
// Add to the User Panel
userPanel.add(new JLabel("First Name:"));
userPanel.add(firstName);
userPanel.add(new JLabel("LastName:"));
userPanel.add(lastName);
userPanel.add(new JLabel("Date of Birth:"));
userPanel.add(dateOfBirth);
userPanel.add(new JLabel("User ID:"));
userPanel.add(userId);
userPanel.add(new JLabel("Borrow Limit:"));
userPanel.add(limit);
userPanel.add(new JLabel("Staff Salary:"));
userPanel.add(salary);
userPanel.add(addStaff);
userPanel.add(addBorrower);
userPanel.add(uStatus);

// Add to the Catalog Panel
catalogPanel.add(new JLabel("Book ISBN:"));
catalogPanel.add(isbn);
catalogPanel.add(new JLabel("Book Title:"));
catalogPanel.add(bookTitle);
catalogPanel.add(new JLabel("Author(s):"));
catalogPanel.add(author);
catalogPanel.add(new JLabel("Subject(s):"));
catalogPanel.add(subject);
catalogPanel.add(new JLabel("Library Book ID:"));
catalogPanel.add(libId);
catalogPanel.add(new JLabel("Cost:"));
catalogPanel.add(cost);
catalogPanel.add(new JLabel("Section"));
catalogPanel.add(section);
catalogPanel.add(new JLabel("Library Copy ID"));
catalogPanel.add(copyId);
catalogPanel.add(addCopy);
catalogPanel.add(removeCopy);
catalogPanel.add(cStatus);

// Add to the Stock Panel
stockPanel.add(new JLabel("Operator ID:"));
stockPanel.add(staffIdQ);
stockPanel.add(new JLabel("Requestor ID:"));
stockPanel.add(borrowerIdQ);
stockPanel.add(new JLabel("Copy ID:"));
```

```
        stockPanel.add(copyIdQ);
        stockPanel.add(new JLabel("Book ID:"));
        stockPanel.add(bookIdQ);
        stockPanel.add(new JLabel("Author:"));
        stockPanel.add(authorQ);
        stockPanel.add(new JLabel("Subject"));
        stockPanel.add(subjectQ);

        stockPanel.add(borrowCopy);
        stockPanel.add(returnCopy);
        stockPanel.add(findAuthor);
        stockPanel.add(findSubject);
        stockPanel.add(whatBorrowedBy);
        stockPanel.add(whoBorrowed);

        stockPanel.add(sStatus);

        uStatus.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
        cStatus.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
        sStatus.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
}

 // Implementation of ActionListener interface.
 public void actionPerformed(ActionEvent event)
     {
         LibraryResultCode rslt=new LibraryResultCode();
         String output = "Ok";

         if (event.getSource() == addStaff)
         {
                 backend.addStaff (_eSystem._lString(firstName.getText()),   'a',
                                   _eSystem._lString(lastName.getText()),    'a',
                                   _eSystem._lString(dateOfBirth.getText()), 'a',
                                   _eSystem._lString(userId.getText()),      'a',
                                   _eSystem._lString(salary.getText()),      'a',
                                   rslt);
                 setResult(rslt.value,output,uStatus);
         }
         else if (event.getSource() == addBorrower)
         {
                 backend.addBorrower(_eSystem._lString(firstName.getText()),  'a',
                                     _eSystem._lString(lastName.getText()),   'a',
                                     _eSystem._lString(dateOfBirth.getText()),'a',
                                     _eSystem._lString(userId.getText()),     'a',
                                     _eSystem._lString(limit.getText()),      'a',
                                     rslt);
                 setResult(rslt.value,output,uStatus);
         }
         else if (event.getSource() == addCopy)
         {
                 backend.addLibraryItem (_eSystem._lString(copyId.getText()),   'a',
                                         _eSystem._lString(libId.getText()),    'a',
                                         _eSystem._lString(cost.getText()),     'a',
                                         _eSystem._lString(section.getText()),  'a',
                                         _eSystem._lString(isbn.getText()),     'a',
                                         _eSystem._lString(bookTitle.getText()),'a',
                                         _eSystem._lString(author.getText()),   'a',
                                         _eSystem._lString(subject.getText()),  'a',
                                         rslt);
                 setResult(rslt.value,output,cStatus);
         }
```

```
    else if (event.getSource() == removeCopy)
    {
            backend.removeLibraryItem (_eSystem._lString(copyId.getText()),'a',
                                       rslt);
            setResult(rslt.value,output,cStatus);
    }
    else if (event.getSource() == borrowCopy)
    {
            backend.borrowLibraryItem (_eSystem._lString(copyIdQ.getText()),    'a',
                                       _eSystem._lString(borrowerIdQ.getText()),'a',
                                       rslt);
            setResult(rslt.value,output,sStatus);
    }
    else if (event.getSource() == returnCopy)
    {
            backend.returnLibraryItem (_eSystem._lString(copyIdQ.getText()),'a',
                                       rslt);
            setResult(rslt.value,output,sStatus);
    }
    else if (event.getSource() == findAuthor)
    {
            _eWrapper__eAny books = new _eWrapper__eAny();
            backend.findItemsByAuthor (_eSystem._lString(authorQ.getText()),'a',
                                       books, new _eSeq(),                  'a',
                                       rslt);
            output = _eSystem._lJavaString((_eSeq)books.value);
            setResult(rslt.value,output,sStatus);

    }
    else if (event.getSource() == findSubject)
    {
            _eWrapper__eAny books = new _eWrapper__eAny();
            backend.findItemsBySubject (_eSystem._lString(subjectQ.getText()),'a',
                                       books, new _eSeq(),                  'a',
                                       rslt);
           output = _eSystem._lJavaString((_eSeq)books.value);
           setResult(rslt.value,output,sStatus);
    }
    else if (event.getSource() == whatBorrowedBy)
    {
    }
    else if (event.getSource() == whoBorrowed)
    {
    }
    else sStatus.setText("Unknown event or return code");
}

public void setResult(int rslt,String value, JLabel status)
{
    if (rslt == LibraryResultCode.success)
    {
        status.setText(value+"Successful operation");
   }
    else if (rslt == LibraryResultCode.unauthorized)
    {
        status.setText(value+"Unauthorized Requestor");
    }
    else if (rslt == LibraryResultCode.unregistered)
    {
        status.setText(value+"Unregistered borrower");
    }
```

```
        else if (rslt == LibraryResultCode.notOwned)
        {
             status.setText(value+"Copy not owned by Library");
        }
        else if (rslt == LibraryResultCode.notAvailable)
        {
            status.setText(value+"Copy checked out");
        }
        else if (rslt == LibraryResultCode.available)
        {
            status.setText(value+"Copy available");
        }
        else if (rslt == LibraryResultCode.unknownAuthor)
        {
            status.setText("No books by this author");
        }
        else if (rslt == LibraryResultCode.unknownSubject)
        {
            status.setText("No books on this subject");
        }
        else if (rslt == LibraryResultCode.maxLimits)
        {
            status.setText(value+"User has reached their borrowing limit");
        }
        else if (rslt == LibraryResultCode.duplicateCopy)
        {
            status.setText(value+ "Copy already exists in Library");
        }
        else if (rslt == LibraryResultCode.duplicateBook)
        {
            status.setText(value+ "Book already exists in Library");
        }
        else if (rslt == LibraryResultCode.noCopy)
        {
            status.setText(value + "Copy is not in Library Database");
        }
        else if (rslt == LibraryResultCode.knownUser)
        {
            status.setText(value + "User already exists");
        }
        else if (rslt == LibraryResultCode.neverBeenReturned)
        {
            status.setText(value + "Copy was never returned");
        }
        else
        {
            status.setText("Unknown result code");
        }
}

private boolean isInteger(String input){
        boolean b=input.length()>0;
        if(b){
            for(int i=0;i<input.length();i++)
                    b=b&&(input.charAt(i)>='0' && input.charAt(i)<='9');
        }
        return b;
}
// main method
public static void main(String jargs[])
{
```

```
        //--------------------------------------------------------------------------
        // Parse arguments - note that the path to the 'running' class file is not
        // available, so the best we can do is provide the directory where the JVM
        // was started ...
        //--------------------------------------------------------------------------
        String pathToClass = System.getProperty("user.dir");
        if (pathToClass.length() != 0 &&
            pathToClass.charAt(pathToClass.length() - 1) != File.separatorChar)
        {
            pathToClass = pathToClass + File.separatorChar;
        }
        Environment context = new Environment(pathToClass);

        try{
            UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
        } catch(Exception e) {}
        LibraryAccess applicationObjectb = new LibraryAccess(context);
    }
}

// End
```

## E.8 LibraryBookDescription.java

```
package oolibrary;


import Ertsys.*;

// Packages imported
import oolibrary.*;


public class LibraryBookDescription extends BookDescription
{
    protected int libraryId;
    protected int cost;
    protected _eSeq section;
    public LibraryBookDescription (int _vlibraryId, int _vcost, _eSeq _vsection, char
        _t0_vsection, int isb, _eSeq titl, char _t0titl, _eSet auth, Author _t0auth, _eSet
        subj, Subject _t0subj)
    {
        super (isb, titl, (char) 0, auth, (Author) null, subj, (Subject) null);
        libraryId = _vlibraryId;
        cost = _vcost;
        section = _vsection;
    }

    public LibraryBookDescription (int _vlibraryId, int _vcost, _eSeq _vsection, char
        _t0_vsection, int isb, _eSeq titl, char _t0titl)
    {
        super (isb, titl, (char) 0);
        libraryId = _vlibraryId;
        cost = _vcost;
        section = _vsection;
    }

    public int getId ()
    {
        return libraryId;
```

```
    }

    public _eSeq _rtoString ()
    {
        return myName ()._oPlusPlus (_eSystem._lString (","), (_eTemplate_0) null).
            _oPlusPlus (_eSystem._ltoString (libraryId), (_eTemplate_0) null);
    }

    public _eSeq getSection ()
    {
        return section;
    }

    public boolean _lEqual (LibraryBookDescription _vArg_12_9)
    {
        if (this == _vArg_12_9) return true;
        if (!super._lEqual (_vArg_12_9)) return false;
        return (((_vArg_12_9.cost == cost) && _vArg_12_9.section._lEqual (section)) && (
            _vArg_12_9.libraryId == libraryId));
    }

    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () ==
            LibraryBookDescription.class && _lEqual ((LibraryBookDescription) _lArg));
    }

}


// End of file.
```

# E.9 LibraryCatalog.java

```
package oolibrary;


import Ertsys.*;

// Packages imported
import oolibrary.*;


public class LibraryCatalog extends _eAny
{
    final void _lc_LibraryCatalog (String _lArg)
    {
        if (_eSystem.enableClassInvariantItem && _eSystem.currentCheckNesting <= _eSystem.
            maxCheckNesting)
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                boolean _vQuantifierResult_16_15;
                {
                    _vQuantifierResult_16_15 = true;
                    int _vCaptureCount_item1_16_35 = allLibraryItems ()._oHash ();
                    int _vLoopCounter_16_22 = 0;
                    for (;;)
                    {
```

```
                if (((_vLoopCounter_16_22 == _vCaptureCount_item1_16_35) || (!
                    _vQuantifierResult_16_15))) break;
                boolean _vQuantifierResult_16_28;
                {
                    _vQuantifierResult_16_28 = true;
                    int _vCaptureCount_item2_16_35 = allLibraryItems ()._oHash ();
                    int _vLoopCounter_16_28 = 0;
                    for (;;)
                    {
                        if (((_vLoopCounter_16_28 == _vCaptureCount_item2_16_35)
                            || (!_vQuantifierResult_16_28))) break;
                        _vQuantifierResult_16_28 = (!(!((LibraryItem)
                            allLibraryItems ()._oIndex (_vLoopCounter_16_22)).
                            _lEqual (((LibraryItem) allLibraryItems ()._oIndex (
                            _vLoopCounter_16_28)))) || (!((LibraryItem)
                            allLibraryItems ()._oIndex (_vLoopCounter_16_22)).
                            getId () == ((LibraryItem) allLibraryItems ()._oIndex
                            (_vLoopCounter_16_28)).getId ())));
                        if ((!_vQuantifierResult_16_28))
                        {
                        }
                        else
                        {
                            _vLoopCounter_16_28 = _eSystem._oSucc (
                                _vLoopCounter_16_28);
                        }
                    }
                }
                _vQuantifierResult_16_15 = _vQuantifierResult_16_28;
                if ((!_vQuantifierResult_16_15))
                {
                }
                else
                {
                    _vLoopCounter_16_22 = _eSystem._oSucc (_vLoopCounter_16_22);
                }
            }
        }
        if (!(_vQuantifierResult_16_15)) throw new _xClassInvariantItem (
            "LibraryCatalog.pd:16,15", _lArg);
    }
    catch (_xCannotEvaluate _lException)
    {
    }
    _eSystem.currentCheckNesting --;
}
if (_eSystem.enableClassInvariantItem && _eSystem.currentCheckNesting <= _eSystem.
    maxCheckNesting)
{
    _eSystem.currentCheckNesting ++;
    try
    {
        if (!((alltheBooks ()._oHash () <= allLibraryItems ()._oHash ()))) throw
            new _xClassInvariantItem ("LibraryCatalog.pd:17,28", _lArg);
    }
    catch (_xCannotEvaluate _lException)
    {
    }
    _eSystem.currentCheckNesting --;
}
if (_eSystem.enableClassInvariantItem && _eSystem.currentCheckNesting <= _eSystem.
```

```
        maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            boolean _vQuantifierResult_18_15;
            {
                _eSet _vCaptureBound_book_18_28 = alltheBooks ();
                _vQuantifierResult_18_15 = true;
                int _vCaptureCount_book_18_28 = _vCaptureBound_book_18_28._oHash ();
                int _vLoopCounter_18_22 = 0;
                for (;;)
                {
                    if (((_vLoopCounter_18_22 == _vCaptureCount_book_18_28) || (!
                        _vQuantifierResult_18_15))) break;
                    _eSet _vChoose_18_44 = null;
                    {
                        _vChoose_18_44 = new _eSet ();
                        int _vCaptureCount_copy_18_57 = allLibraryItems ()._oHash ();
                        int _vLoopCounter_18_51 = 0;
                        for (;;)
                        {
                            if ((_vLoopCounter_18_51 == _vCaptureCount_copy_18_57))
                                break;
                            if (((LibraryBookDescription) _vCaptureBound_book_18_28.
                                _oIndex (_vLoopCounter_18_22))._lEqual (((LibraryItem)
                                allLibraryItems ()._oIndex (_vLoopCounter_18_51)).
                                getBook ()))
                            {
                                _vChoose_18_44 = _vChoose_18_44.append (
                                    allLibraryItems ()._oIndex (_vLoopCounter_18_51));
                            }
                            else
                            {
                            }
                            _vLoopCounter_18_51 = _eSystem._oSucc (_vLoopCounter_18_51
                                );
                        }
                    }
                    _vQuantifierResult_18_15 = (_vChoose_18_44._oHash () <= 1000);
                    if ((!_vQuantifierResult_18_15))
                    {
                    }
                    else
                    {
                        _vLoopCounter_18_22 = _eSystem._oSucc (_vLoopCounter_18_22);
                    }
                }
            }
            if (!(_vQuantifierResult_18_15)) throw new _xClassInvariantItem (
                "LibraryCatalog.pd:18,15", _lArg);
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
}

void _lClassInvariantCheck (String _lArg)
{
```

```
    _lc_LibraryCatalog (_lArg);
}

private _eSet allLibraryItems ()
{
    return allLibItems.ran ();
}

protected _eSet alltheBooks ()
{
    _eSet _vForYield_14_12;
    {
        _vForYield_14_12 = new _eSet ();
        int _vCaptureCount_item_14_22 = allLibraryItems ()._oHash ();
        int _vLoopCounter_14_16 = 0;
        for (;;)
        {
            if ((_vLoopCounter_14_16 == _vCaptureCount_item_14_22)) break;
            _vForYield_14_12 = _vForYield_14_12.append (((_eAny) ((LibraryItem)
                allLibraryItems ()._oIndex (_vLoopCounter_14_16)).getBook ()));
            _vLoopCounter_14_16 = _eSystem._oSucc (_vLoopCounter_14_16);
        }
    }
    return _vForYield_14_12;
}

private _n1_HashedBucket allLibItems;
public LibraryCatalog ()
{
    super ();
    allLibItems = new _n1_HashedBucket (1000);
    _lc_LibraryCatalog ("LibraryCatalog.pd:34,9");
}

public boolean newItemId (int id)
{
    boolean _vQuantifierResult_37_12;
    {
        _eSet _vCaptureBound_items_37_26 = allKnownItems ();
        _vQuantifierResult_37_12 = true;
        int _vCaptureCount_items_37_26 = _vCaptureBound_items_37_26._oHash ();
        int _vLoopCounter_37_19 = 0;
        for (;;)
        {
            if (((_vLoopCounter_37_19 == _vCaptureCount_items_37_26) || (!
                _vQuantifierResult_37_12))) break;
            _vQuantifierResult_37_12 = (!(id == ((LibraryItem)
                _vCaptureBound_items_37_26._oIndex (_vLoopCounter_37_19)).getId ()));
            if ((!_vQuantifierResult_37_12))
            {
            }
            else
            {
                _vLoopCounter_37_19 = _eSystem._oSucc (_vLoopCounter_37_19);
            }
        }
    }
    return _vQuantifierResult_37_12;
}

public boolean newBook (LibraryBookDescription b)
```

```
{
    return allKnownBooks ()._ovIn (((_eAny) b));
}

public _eSet allKnownItems ()
{
    return allLibItems.ran ();
}

public _eSet allKnownBooks ()
{
    _eSet _vForYield_51_19;
    {
        _eSet _vCaptureBound_item_51_29 = allKnownItems ();
        _vForYield_51_19 = new _eSet ();
        int _vCaptureCount_item_51_29 = _vCaptureBound_item_51_29._oHash ();
        int _vLoopCounter_51_23 = 0;
        for (;;)
        {
            if ((_vLoopCounter_51_23 == _vCaptureCount_item_51_29)) break;
            _vForYield_51_19 = _vForYield_51_19.append (((_eAny) ((LibraryItem)
                _vCaptureBound_item_51_29._oIndex (_vLoopCounter_51_23)).getBook ()));
            _vLoopCounter_51_23 = _eSystem._oSucc (_vLoopCounter_51_23);
        }
    }
    return _vForYield_51_19;
}

public _eSet allKnownItemIds ()
{
    _eSet _vForYield_55_12;
    {
        _eSet _vCaptureBound_item_55_22 = allKnownItems ();
        _vForYield_55_12 = new _eSet ();
        int _vCaptureCount_item_55_22 = _vCaptureBound_item_55_22._oHash ();
        int _vLoopCounter_55_16 = 0;
        for (;;)
        {
            if ((_vLoopCounter_55_16 == _vCaptureCount_item_55_22)) break;
            _vForYield_55_12 = _vForYield_55_12.append (((_eAny) new _eWrapper_int (((
                LibraryItem) _vCaptureBound_item_55_22._oIndex (_vLoopCounter_55_16)).
                getId ())));
            _vLoopCounter_55_16 = _eSystem._oSucc (_vLoopCounter_55_16);
        }
    }
    return _vForYield_55_12;
}

public _eSet allKnownBookIds ()
{
    _eSet _vForYield_58_12;
    {
        _eSet _vCaptureBound_book_58_22 = allKnownBooks ();
        _vForYield_58_12 = new _eSet ();
        int _vCaptureCount_book_58_22 = _vCaptureBound_book_58_22._oHash ();
        int _vLoopCounter_58_16 = 0;
        for (;;)
        {
            if ((_vLoopCounter_58_16 == _vCaptureCount_book_58_22)) break;
            _vForYield_58_12 = _vForYield_58_12.append (((_eAny) new _eWrapper_int (((
                LibraryBookDescription) _vCaptureBound_book_58_22._oIndex (
```

```
                _vLoopCounter_58_16)).getId ()));
            _vLoopCounter_58_16 = _eSystem._oSucc (_vLoopCounter_58_16);
        }
    }
    return _vForYield_58_12;
}

public boolean hasItem (LibraryItem copy)
{
    return allLibItems._ovIn (((_eAny) copy));
}

public boolean uniqueItemId (int id)
{
    return (!allKnownItemIds ()._ovIn (((_eAny) new _eWrapper_int (id))));
}

public void addLibraryItem (LibraryItem copy)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!((!hasItem (copy)))) throw new _xPre ("LibraryCatalog.pd:70,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(uniqueItemId (copy.getId ()))) throw new _xPre (
                "LibraryCatalog.pd:71,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    _n1_HashedBucket _vUnshare_74_13 = ((_n1_HashedBucket) allLibItems._lClone ());
    allLibItems = _vUnshare_74_13;
    _vUnshare_74_13.add (((_eAny) copy));
    _lClassInvariantCheck ("LibraryCatalog.pd:75,9");
}

public void removeLibraryItem (LibraryItem copy)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(hasItem (copy))) throw new _xPre ("LibraryCatalog.pd:78,13");
```

```
            }
            catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
        _n1_HashedBucket _vUnshare_80_13 = ((_n1_HashedBucket) allLibItems._lClone ());
        allLibItems = _vUnshare_80_13;
        _vUnshare_80_13.remove (((_eAny) copy));
        _lClassInvariantCheck ("LibraryCatalog.pd:81,9");
    }

    public boolean _lEqual (LibraryCatalog _vArg_11_9)
    {
        if (this == _vArg_11_9) return true;
        return _vArg_11_9.allLibraryItems ()._lEqual (allLibraryItems ());
    }

    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () == LibraryCatalog.
            class && _lEqual ((LibraryCatalog) _lArg));
    }

}


// End of file.
```

# E.10 LibraryDB.java

```
package oolibrary;


import Ertsys.*;

// Packages imported
import oolibrary.*;


public class LibraryDB extends _eAny
{
    protected LibraryStock books;
    protected UserBase users;
    protected _eSeq today;
    public LibraryDB (Environment context)
    {
        super ();
        today = context.getCurrentDateTime ()._rtoString ();
        books = new LibraryStock ();
        users = new UserBase ();
    }

    public boolean isNumber (_eSeq val, char _t0val)
    {
        boolean _vQuantifierResult_23_12;
        {
            _vQuantifierResult_23_12 = true;
            int _vCaptureCount_i_23_26 = val._oHash ();
            int _vLoopCounter_23_19 = 0;
```

```
        for (;;)
        {
            if (((_vCaptureCount_i_23_26 <= _vLoopCounter_23_19) || (!
                _vQuantifierResult_23_12))) break;
            _vQuantifierResult_23_12 = (_eSystem._lisDigit (((_eWrapper_char) val.
                _oIndex (_vLoopCounter_23_19)).value) && (0 < val._oHash ()));
            if ((!_vQuantifierResult_23_12))
            {
            }
            else
            {
                _vLoopCounter_23_19 = _eSystem._oSucc (_vLoopCounter_23_19);
            }
        }
    }
    return _vQuantifierResult_23_12;
}

public int getNumber (_eSeq val, char _t0val)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(isNumber (val, (char) 0))) throw new _xPre ("LibraryDB.pd:26,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    return _eSystem._lstringToNat (val, (char) 0);
}

public _eSeq firstComma (_eSeq s, char _t0s)
{
    int _vLet_n_31_14 = s.findFirst (((_eAny) new _eWrapper_char (','')));
    return ((_vLet_n_31_14 < 0) ?
    s :
    s.take ((1 + _vLet_n_31_14)));
}

public _eSet makeAuthorSet (_eSeq s, char _t0s)
{
    _eSeq _vLet_stripped_38_18 = s;
    if (_vLet_stripped_38_18.empty ())
    {
        return new _eSet ();
    }
    else
    {
        _eSeq _vLet_a_41_20 = firstComma (_vLet_stripped_38_18, (char) 0);
        return makeAuthorSet (_vLet_stripped_38_18.drop (_vLet_a_41_20._oHash ()), (
            char) 0).append (((_eAny) new Author (_vLet_a_41_20, (char) 0)));
    }
}

public _eSet makeSubjectSet (_eSeq s, char _t0s)
{
```

```
    _eSeq _vLet_stripped_49_18 = s;
    if (_vLet_stripped_49_18.empty ())
    {
        return new _eSet ();
    }
    else
    {
        _eSeq _vLet_su_52_20 = firstComma (_vLet_stripped_49_18, (char) 0);
        return makeSubjectSet (_vLet_stripped_49_18.drop (_vLet_su_52_20._oHash ()), (
            char) 0).append (((_eAny) new Subject (_vLet_su_52_20, (char) 0)));
    }
}

public LibraryBookDescription makeLibraryBook (_eSeq libID, char _t0libID, _eSeq cost,
    char _t0cost, _eSeq section, char _t0section, _eSeq isbn, char _t0isbn, _eSeq
    title, char _t0title, _eSeq authors, char _t0authors, _eSeq subjects, char
    _t0subjects)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(isNumber (libID, (char) 0))) throw new _xPre ("LibraryDB.pd:59,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(isNumber (cost, (char) 0))) throw new _xPre ("LibraryDB.pd:59,30");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(isNumber (isbn, (char) 0))) throw new _xPre ("LibraryDB.pd:59,46");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    _eSet _vLet_authset_60_16 = makeAuthorSet (authors, (char) 0);
    _eSet _vLet_subjset_60_53 = makeSubjectSet (subjects, (char) 0);
    return new LibraryBookDescription (getNumber (libID, (char) 0), getNumber (cost, (
        char) 0), section, (char) 0, getNumber (isbn, (char) 0), title, (char) 0,
```

```
        _vLet_authset_60_16, (Author) null, _vLet_subjset_60_53, (Subject) null);
}

public Staff makeStaffMember (_eSeq first, char _t0first, _eSeq last, char _t0last,
    _eSeq doB, char _t0doB, _eSeq id, char _t0id, _eSeq salary, char _t0salary)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(isNumber (id, (char) 0))) throw new _xPre ("LibraryDB.pd:65,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(isNumber (salary, (char) 0))) throw new _xPre ("LibraryDB.pd:65,26")
                ;
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    return new Staff (first, (char) 0, last, (char) 0, doB, (char) 0, getNumber (id, (
        char) 0), getNumber (salary, (char) 0));
}

public Borrower makeBorrowerMember (_eSeq first, char _t0first, _eSeq last, char
    _t0last, _eSeq doB, char _t0doB, _eSeq id, char _t0id, _eSeq limit, char _t0limit)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(isNumber (id, (char) 0))) throw new _xPre ("LibraryDB.pd:69,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(isNumber (limit, (char) 0))) throw new _xPre ("LibraryDB.pd:69,26");
        }
```

```
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    return new Borrower (first, (char) 0, last, (char) 0, doB, (char) 0, getNumber (id
        , (char) 0), getNumber (limit, (char) 0));
}

public LibraryItem makeLibraryItem (_eSeq id, char _t0id, _eSeq libID, char _t0libID,
    _eSeq cost, char _t0cost, _eSeq section, char _t0section, _eSeq isbn, char _t0isbn
    , _eSeq title, char _t0title, _eSeq authors, char _t0authors, _eSeq subjects, char
    _t0subjects)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(isNumber (id, (char) 0))) throw new _xPre ("LibraryDB.pd:73,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(isNumber (libID, (char) 0))) throw new _xPre ("LibraryDB.pd:73,26");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(isNumber (cost, (char) 0))) throw new _xPre ("LibraryDB.pd:73,43");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(isNumber (isbn, (char) 0))) throw new _xPre ("LibraryDB.pd:73,59");
        }
```

```
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    LibraryBookDescription _vLet_libBook_74_17 = makeLibraryBook (libID, (char) 0,
        cost, (char) 0, section, (char) 0, isbn, (char) 0, title, (char) 0, authors, (
        char) 0, subjects, (char) 0);
    return new LibraryItem (getNumber (id, (char) 0), today, (char) 0,
        _vLet_libBook_74_17);
}

public LibraryItem makeLibraryItem (_eSeq id, char _t0id, LibraryBookDescription
    libBook)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(isNumber (id, (char) 0))) throw new _xPre ("LibraryDB.pd:79,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    return new LibraryItem (getNumber (id, (char) 0), today, (char) 0, libBook);
}

public void addStaff (_eSeq first, char _t0first, _eSeq last, char _t0last, _eSeq doB,
    char _t0doB, _eSeq id, char _t0id, _eSeq salary, char _t0salary, LibraryResultCode
    rslt)
{
    if (((!isNumber (id, (char) 0)) || (!isNumber (salary, (char) 0))))
    {
        rslt.value = LibraryResultCode.incorrectInput;
    }
    else
    {
        Staff _vLet_s_85_23 = makeStaffMember (first, (char) 0, last, (char) 0, doB, (
            char) 0, id, (char) 0, salary, (char) 0);
        if (users.uniqueId (_vLet_s_85_23))
        {
            UserBase _vUnshare_86_37 = ((UserBase) users._lClone ());
            users = _vUnshare_86_37;
            _vUnshare_86_37.addStaff (_vLet_s_85_23);
            rslt.value = LibraryResultCode.success;
        }
        else
        {
            rslt.value = LibraryResultCode.knownUser;
        }
    }
}

public void addBorrower (_eSeq first, char _t0first, _eSeq last, char _t0last, _eSeq
    doB, char _t0doB, _eSeq id, char _t0id, _eSeq limit, char _t0limit,
    LibraryResultCode rslt)
{
```

```
    if (((!isNumber (id, (char) 0)) || (!isNumber (limit, (char) 0))))
    {
        rslt.value = LibraryResultCode.incorrectInput;
    }
    else
    {
        Borrower _vLet_b_95_22 = makeBorrowerMember (first, (char) 0, last, (char) 0,
            doB, (char) 0, id, (char) 0, limit, (char) 0);
        if (users.uniqueId (_vLet_b_95_22))
        {
            UserBase _vUnshare_96_37 = ((UserBase) users._lClone ());
            users = _vUnshare_96_37;
            _vUnshare_96_37.addBorrower (_vLet_b_95_22);
            rslt.value = LibraryResultCode.success;
        }
        else
        {
            rslt.value = LibraryResultCode.knownUser;
        }
    }
}

public void nextDay (_eSeq next, char _t0next)
{
    today = next;
}

public void addLibraryItem (_eSeq id, char _t0id, _eSeq libID, char _t0libID, _eSeq
    cost, char _t0cost, _eSeq section, char _t0section, _eSeq isbn, char _t0isbn,
    _eSeq title, char _t0title, _eSeq authors, char _t0authors, _eSeq subjects, char
    _t0subjects, LibraryResultCode rslt)
{
    if (((((!isNumber (id, (char) 0)) || (!isNumber (libID, (char) 0))) || (!isNumber
        (cost, (char) 0))) || (!isNumber (isbn, (char) 0))))
    {
        rslt.value = LibraryResultCode.incorrectInput;
    }
    else
    {
        LibraryItem _vLet_newCopy_109_22 = makeLibraryItem (id, (char) 0, libID, (char
            ) 0, cost, (char) 0, section, (char) 0, isbn, (char) 0, title, (char) 0,
            authors, (char) 0, subjects, (char) 0);
        if (books.hasItem (_vLet_newCopy_109_22))
        {
            rslt.value = LibraryResultCode.duplicateCopy;
        }
        else
        {
            LibraryStock _vUnshare_111_21 = ((LibraryStock) books._lClone ());
            books = _vUnshare_111_21;
            _vUnshare_111_21.addLibraryItem (_vLet_newCopy_109_22);
            rslt.value = LibraryResultCode.success;
        }
    }
}

public void removeLibraryItem (_eSeq id, char _t0id, LibraryResultCode rslt)
{
    if ((!isNumber (id, (char) 0)))
    {
        rslt.value = LibraryResultCode.incorrectInput;
```

```
    }
    else
    {
        int _vLet_copyId_119_22 = getNumber (id, (char) 0);
        if ((!books.hasId (_vLet_copyId_119_22)))
        {
            rslt.value = LibraryResultCode.notOwned;
        }
        else
        {
            LibraryItem _vLet_newCopy_121_27 = books.findItem (_vLet_copyId_119_22);
            if ((!books.itemAvailable (_vLet_newCopy_121_27)))
            {
                rslt.value = LibraryResultCode.notAvailable;
            }
            else
            {
                LibraryStock _vUnshare_123_26 = ((LibraryStock) books._lClone ());
                books = _vUnshare_123_26;
                _vUnshare_123_26.removeLibraryItem (_vLet_newCopy_121_27);
                rslt.value = LibraryResultCode.success;
            }
        }
    }
}

public void borrowLibraryItem (_eSeq itemId, char _t0itemId, _eSeq borrowerId, char
    _t0borrowerId, LibraryResultCode rslt)
{
    if (((!isNumber (itemId, (char) 0)) || (!isNumber (borrowerId, (char) 0))))
    {
        rslt.value = LibraryResultCode.incorrectInput;
    }
    else
    {
        int _vLet_copyId_132_22 = getNumber (itemId, (char) 0);
        int _vLet_borrId_132_53 = getNumber (borrowerId, (char) 0);
        if ((!books.hasId (_vLet_copyId_132_22)))
        {
            rslt.value = LibraryResultCode.notOwned;
        }
        else if ((!users.hasId (_vLet_borrId_132_53)))
        {
            rslt.value = LibraryResultCode.unregistered;
        }
        else
        {
            LibraryItem _vLet_copy_135_26 = books.findItem (_vLet_copyId_132_22);
            Borrower _vLet_borrower_135_60 = users.findUser (_vLet_borrId_132_53);
            if ((!books.itemAvailable (_vLet_copy_135_26)))
            {
                rslt.value = LibraryResultCode.notAvailable;
            }
            else if ((!books.withinLimits (_vLet_borrower_135_60)))
            {
                rslt.value = LibraryResultCode.maxLimits;
            }
            else
            {
                LibraryStock _vUnshare_138_26 = ((LibraryStock) books._lClone ());
                books = _vUnshare_138_26;
```

```
                _vUnshare_138_26.borrowLibraryItem (_vLet_copy_135_26,
                    _vLet_borrower_135_60, today, (char) 0);
                rslt.value = LibraryResultCode.success;
            }
        }
    }
}

public void returnLibraryItem (_eSeq itemId, char _t0itemId, LibraryResultCode rslt)
{
    if ((!isNumber (itemId, (char) 0)))
    {
        rslt.value = LibraryResultCode.incorrectInput;
    }
    else
    {
        int _vLet_copyId_147_22 = getNumber (itemId, (char) 0);
        if ((!books.hasId (_vLet_copyId_147_22)))
        {
            rslt.value = LibraryResultCode.notOwned;
        }
        else
        {
            LibraryItem _vLet_copy_149_26 = books.findItem (_vLet_copyId_147_22);
            if (books.itemAvailable (_vLet_copy_149_26))
            {
                rslt.value = LibraryResultCode.available;
            }
            else
            {
                LibraryStock _vUnshare_151_26 = ((LibraryStock) books._lClone ());
                books = _vUnshare_151_26;
                _vUnshare_151_26.returnLibraryItem (_vLet_copy_149_26);
                rslt.value = LibraryResultCode.success;
            }
        }
    }
}

public void findItemsByAuthor (_eSeq author, char _t0author, _eWrapper__eAny booksList
    , _eSeq _t0booksList, char _t1booksList, LibraryResultCode rslt)
{
    _eSet _vLet_authset_158_20 = makeAuthorSet (author, (char) 0);
    if (_vLet_authset_158_20.empty ())
    {
        booksList.value = _eSystem._lString ("No Author");
        rslt.value = LibraryResultCode.incorrectInput;
    }
    else
    {
        _eSet _vLet_bookset_161_22 = books.findBookByAuthor (_vLet_authset_158_20, (
            Author) null);
        booksList.value = _vLet_bookset_161_22._rtoString ();
        rslt.value = LibraryResultCode.success;
    }
}

public void findItemsBySubject (_eSeq subject, char _t0subject, _eWrapper__eAny
    booksList, _eSeq _t0booksList, char _t1booksList, LibraryResultCode rslt)
{
    _eSet _vLet_subjset_168_20 = makeSubjectSet (subject, (char) 0);
```

```
        if (_vLet_subjset_168_20.empty ())
        {
            booksList.value = _eSystem._lString ("No Subject");
            rslt.value = LibraryResultCode.incorrectInput;
        }
        else
        {
            _eSet _vLet_bookset_171_22 = books.findBookBySubject (_vLet_subjset_168_20, (
                Subject) null);
            booksList.value = _vLet_bookset_171_22._rtoString ();
            rslt.value = LibraryResultCode.success;
        }
    }

    public boolean _lEqual (LibraryDB _vArg_11_9)
    {
        if (this == _vArg_11_9) return true;
        return ((_vArg_11_9.users._lEqual (users) && _vArg_11_9.today._lEqual (today)) &&
            _vArg_11_9.books._lEqual (books));
    }

    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () == LibraryDB.class &&
            _lEqual ((LibraryDB) _lArg));
    }

}


// End of file.
```

# E.11 LibraryItem.java

```
package oolibrary;


import Ertsys.*;

// Packages imported
import oolibrary.*;


public class LibraryItem extends _eAny
{
    protected int copyId;
    protected _eSeq dateAcquired;
    protected LibraryBookDescription myLibraryBookDescription;
    public LibraryItem (int _vcopyId, _eSeq _vdateAcquired, char _t0_vdateAcquired,
        LibraryBookDescription _vmyLibraryBookDescription)
    {
        super ();
        if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
            )
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                if (!((_vcopyId <= 999))) throw new _xPre ("LibraryItem.pd:17,19");
            }
```

```
            catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
        copyId = _vcopyId;
        dateAcquired = _vdateAcquired;
        myLibraryBookDescription = _vmyLibraryBookDescription;
    }

    public int getId ()
    {
        return ((1000 * myLibraryBookDescription.getId ()) + copyId);
    }

    public LibraryBookDescription getBook ()
    {
        return myLibraryBookDescription;
    }

    public int hash ()
    {
        return getId ();
    }

    public boolean _lEqual (LibraryItem _vArg_11_9)
    {
        if (this == _vArg_11_9) return true;
        return ((_vArg_11_9.dateAcquired._lEqual (dateAcquired) && _vArg_11_9.
            myLibraryBookDescription._lEqual (myLibraryBookDescription)) && (_vArg_11_9.
            copyId == copyId));
    }

    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () == LibraryItem.class
            && _lEqual ((LibraryItem) _lArg));
    }

}


// End of file.
```

## E.12 LibraryResultCode.java

```
package oolibrary;


import Ertsys.*;

// Packages imported
import oolibrary.*;


public final class LibraryResultCode extends _eEnumBase
{
    public LibraryResultCode ()
    {
    }
```

```
    public LibraryResultCode (int _lArg1)
    {
        super (_lArg1);
    }

    public static _eSeq _oRange (int _lArg1, int _lArg2)
    {
        _eSeq _lResult = new _eSeq ();
        while (_lArg1 <= _lArg2) _lResult = _lResult._lAppend (new LibraryResultCode (
            _lArg1 ++));
        return _lResult;
    }

    public static final int success = 0, unauthorized = 1, unregistered = 2, notOwned = 3,
        notAvailable = 4, available = 5, unknownAuthor = 6, unknownSubject = 7, maxLimits
        = 8, duplicateCopy = 9, duplicateBook = 10, noCopy = 11, knownUser = 12,
        neverBeenReturned = 13, incorrectInput = 14;
    private static final String _lnames [] =
    {
        "success", "unauthorized", "unregistered", "notOwned", "notAvailable", "available"
            , "unknownAuthor", "unknownSubject", "maxLimits", "duplicateCopy",
            "duplicateBook", "noCopy", "knownUser", "neverBeenReturned", "incorrectInput"
    };

    public static _eSeq _ltoString (int _larg)
    {
        return _eSystem._lString (_lnames [_larg]);
    }

    public _eSeq _rtoString ()
    {
        return _eSystem._lString (_lnames [value]);
    }

}


// End of file.
```

# E.13 LibraryStock.java

```
package oolibrary;


import Ertsys.*;

// Packages imported
import oolibrary.*;


public class LibraryStock extends _eAny
{
    protected LibraryCatalog catalog;
    private _eSet currentlyBorrowed ()
    {
        return borrowed.ran ();
    }

    private _n1_PriorityQueue borrowed;
```

```
public LibraryStock ()
{
    super ();
    catalog = new LibraryCatalog ();
    borrowed = new _n1_PriorityQueue ();
}

public _eSet checkedOut ()
{
    _eSet _vForYield_33_18;
    {
        _eSet _vCaptureBound_b_33_35 = borrowed.ran ();
        _vForYield_33_18 = new _eSet ();
        int _vCaptureCount_b_33_35 = _vCaptureBound_b_33_35._oHash ();
        int _vLoopCounter_33_23 = 0;
        for (;;)
        {
            if ((_vLoopCounter_33_23 == _vCaptureCount_b_33_35)) break;
            _vForYield_33_18 = _vForYield_33_18.append (((_eAny) ((Borrowing)
                _vCaptureBound_b_33_35._oIndex (_vLoopCounter_33_23)).getItem ()));
            _vLoopCounter_33_23 = _eSystem._oSucc (_vLoopCounter_33_23);
        }
    }
    return _vForYield_33_18;
}

public _eSet available ()
{
    return catalog.allKnownItems ()._oMinusMinus (checkedOut (), (_eTemplate_0) null);
}

public boolean itemAvailable (LibraryItem c)
{
    return available ()._ovIn (((_eAny) c));
}

public _eSet booksHeld ()
{
    return catalog.allKnownBooks ();
}

public boolean itemCurrentlyBorrowed (LibraryItem copy)
{
    return checkedOut ()._ovIn (((_eAny) copy));
}

public Borrowing copyBorrowing (LibraryItem copy)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(itemCurrentlyBorrowed (copy))) throw new _xPre (
                "LibraryStock.pd:49,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
```

```
    }
    Borrowing _vChoose_51_18 = null;
    {
        _eSet _vCaptureBound_b_51_36 = borrowed.ran ();
        boolean _vSelectorCondition_51_18;
        _vSelectorCondition_51_18 = false;
        int _vCaptureCount_b_51_36 = _vCaptureBound_b_51_36._oHash ();
        int _vLoopCounter_51_24 = 0;
        for (;;)
        {
            if (((_vLoopCounter_51_24 == _vCaptureCount_b_51_36) ||
                _vSelectorCondition_51_18)) break;
            _vSelectorCondition_51_18 = ((Borrowing) _vCaptureBound_b_51_36._oIndex (
                _vLoopCounter_51_24)).getItem ()._lEqual (copy);
            if (_vSelectorCondition_51_18)
            {
                _vChoose_51_18 = ((Borrowing) _vCaptureBound_b_51_36._oIndex (
                    _vLoopCounter_51_24));
            }
            else
            {
            }
            if (_vSelectorCondition_51_18)
            {
            }
            else
            {
                _vLoopCounter_51_24 = _eSystem._oSucc (_vLoopCounter_51_24);
            }
        }
        if (_eSystem.enableThatOrAny && _eSystem.currentCheckNesting <= _eSystem.
            maxCheckNesting)
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                if (!(_vSelectorCondition_51_18)) throw new _xThatOrAny (
                    "LibraryStock.pd:51,18");
            }
            catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
    }
    return _vChoose_51_18;
}

public boolean hasItem (LibraryItem copy)
{
    return catalog.hasItem (copy);
}

public boolean hasId (int id)
{
    return catalog.allKnownItemIds ()._ovIn (((_eAny) new _eWrapper_int (id)));
}

public boolean withinLimits (Borrower bor)
{
    return true;
```

```
}

public LibraryItem findItem (int id)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(hasId (id))) throw new _xPre ("LibraryStock.pd:64,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    LibraryItem _vChoose_65_12 = null;
    {
        _eSet _vCaptureBound_copy_65_31 = catalog.allKnownItems ();
        boolean _vSelectorCondition_65_12;
        _vSelectorCondition_65_12 = false;
        int _vCaptureCount_copy_65_31 = _vCaptureBound_copy_65_31._oHash ();
        int _vLoopCounter_65_17 = 0;
        for (;;)
        {
            if (((_vLoopCounter_65_17 == _vCaptureCount_copy_65_31) ||
                _vSelectorCondition_65_12)) break;
            _vSelectorCondition_65_12 = (((LibraryItem) _vCaptureBound_copy_65_31.
                _oIndex (_vLoopCounter_65_17)).getId () == id);
            if (_vSelectorCondition_65_12)
            {
                _vChoose_65_12 = ((LibraryItem) _vCaptureBound_copy_65_31._oIndex (
                    _vLoopCounter_65_17));
            }
            else
            {
            }
            if (_vSelectorCondition_65_12)
            {
            }
            else
            {
                _vLoopCounter_65_17 = _eSystem._oSucc (_vLoopCounter_65_17);
            }
        }
        if (_eSystem.enableThatOrAny && _eSystem.currentCheckNesting <= _eSystem.
            maxCheckNesting)
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                if (!(_vSelectorCondition_65_12)) throw new _xThatOrAny (
                    "LibraryStock.pd:65,12");
            }
            catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
    }
```

```
    return _vChoose_65_12;
}

public _eSet findBookByAuthor (_eSet aut, Author _t0aut)
{
    _eSet _vChoose_68_12 = null;
    {
        _eSet _vCaptureBound_b_68_21 = booksHeld ();
        _vChoose_68_12 = new _eSet ();
        int _vCaptureCount_b_68_21 = _vCaptureBound_b_68_21._oHash ();
        int _vLoopCounter_68_18 = 0;
        for (;;)
        {
            if ((_vLoopCounter_68_18 == _vCaptureCount_b_68_21)) break;
            if (aut._oLessLessEq (((LibraryBookDescription) _vCaptureBound_b_68_21.
                _oIndex (_vLoopCounter_68_18)).getAuthors (), (_eTemplate_0) null))
            {
                _vChoose_68_12 = _vChoose_68_12.append (_vCaptureBound_b_68_21._oIndex
                    (_vLoopCounter_68_18));
            }
            else
            {
            }
            _vLoopCounter_68_18 = _eSystem._oSucc (_vLoopCounter_68_18);
        }
    }
    return _vChoose_68_12;
}

public _eSet findBookBySubject (_eSet sub, Subject _t0sub)
{
    _eSet _vChoose_71_12 = null;
    {
        _eSet _vCaptureBound_b_71_21 = booksHeld ();
        _vChoose_71_12 = new _eSet ();
        int _vCaptureCount_b_71_21 = _vCaptureBound_b_71_21._oHash ();
        int _vLoopCounter_71_18 = 0;
        for (;;)
        {
            if ((_vLoopCounter_71_18 == _vCaptureCount_b_71_21)) break;
            if (sub._oLessLessEq (((LibraryBookDescription) _vCaptureBound_b_71_21.
                _oIndex (_vLoopCounter_71_18)).getSubjects (), (_eTemplate_0) null))
            {
                _vChoose_71_12 = _vChoose_71_12.append (_vCaptureBound_b_71_21._oIndex
                    (_vLoopCounter_71_18));
            }
            else
            {
            }
            _vLoopCounter_71_18 = _eSystem._oSucc (_vLoopCounter_71_18);
        }
    }
    return _vChoose_71_12;
}

public _eSet findCopiesBorrowedByBorrower (Borrower b)
{
    _eSet _vChoose_74_12 = null;
    {
        _eSet _vCaptureBound_item_74_24 = checkedOut ();
        _vChoose_74_12 = new _eSet ();
```

```
        int _vCaptureCount_item_74_24 = _vCaptureBound_item_74_24._oHash ();
        int _vLoopCounter_74_18 = 0;
        for (;;)
        {
            if ((_vLoopCounter_74_18 == _vCaptureCount_item_74_24)) break;
            if (copyBorrowing (((LibraryItem) _vCaptureBound_item_74_24._oIndex (
                _vLoopCounter_74_18))).borrowedBy ()._lEqual (b))
            {
                _vChoose_74_12 = _vChoose_74_12.append (_vCaptureBound_item_74_24.
                    _oIndex (_vLoopCounter_74_18));
            }
            else
            {
            }
            _vLoopCounter_74_18 = _eSystem._oSucc (_vLoopCounter_74_18);
        }
    }
    return _vChoose_74_12;
}

public void borrowLibraryItem (LibraryItem copy, Borrower bor, _eSeq date, char
    _t0date)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(itemAvailable (copy))) throw new _xPre ("LibraryStock.pd:77,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    _n1_PriorityQueue _vUnshare_79_14 = ((_n1_PriorityQueue) borrowed._lClone ());
    borrowed = _vUnshare_79_14;
    _vUnshare_79_14.insert (((_eAny) new Borrowing (copy, bor, date, (char) 0)));
}

public void returnLibraryItem (LibraryItem copy)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(itemCurrentlyBorrowed (copy))) throw new _xPre (
                "LibraryStock.pd:83,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    _n1_PriorityQueue _vUnshare_85_14 = ((_n1_PriorityQueue) borrowed._lClone ());
    borrowed = _vUnshare_85_14;
    _vUnshare_85_14.remove ();
}
```

```
public void addLibraryItem (LibraryItem copy)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!((!hasItem (copy)))) throw new _xPre ("LibraryStock.pd:89,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    LibraryCatalog _vUnshare_90_14 = ((LibraryCatalog) catalog._lClone ());
    catalog = _vUnshare_90_14;
    _vUnshare_90_14.addLibraryItem (copy);
}

public void removeLibraryItem (LibraryItem copy)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(hasItem (copy))) throw new _xPre ("LibraryStock.pd:93,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(itemAvailable (copy))) throw new _xPre ("LibraryStock.pd:94,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    LibraryCatalog _vUnshare_95_14 = ((LibraryCatalog) catalog._lClone ());
    catalog = _vUnshare_95_14;
    _vUnshare_95_14.removeLibraryItem (copy);
}

public boolean _lEqual (LibraryStock _vArg_11_9)
{
    if (this == _vArg_11_9) return true;
    return (_vArg_11_9.catalog._lEqual (catalog) && _vArg_11_9.currentlyBorrowed ().
        _lEqual (currentlyBorrowed ()));
}
```

```
    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () == LibraryStock.class
            && _lEqual ((LibraryStock) _lArg));
    }

}


// End of file.
```

## E.14 Person.java

```
package oolibrary;


import Ertsys.*;

// Packages imported
import oolibrary.*;


public class Person extends _eAny
{
    protected _eSeq firstName;
    protected _eSeq lastName;
    protected _eSeq dateOfBirth;
    public Person (_eSeq _vfirstName, char _t0_vfirstName, _eSeq _vlastName, char
        _t0_vlastName, _eSeq _vdateOfBirth, char _t0_vdateOfBirth)
    {
        super ();
        firstName = _vfirstName;
        lastName = _vlastName;
        dateOfBirth = _vdateOfBirth;
    }

    public _eSeq getName ()
    {
        return firstName._oPlusPlus (_eSystem._lString (" "), (_eTemplate_0) null).
            _oPlusPlus (lastName, (_eTemplate_0) null);
    }

    public _eSeq _rtoString ()
    {
        return getName ();
    }

    public boolean _lEqual (Person _vArg_11_9)
    {
        if (this == _vArg_11_9) return true;
        return ((_vArg_11_9.lastName._lEqual (lastName) && _vArg_11_9.dateOfBirth._lEqual
            (dateOfBirth)) && _vArg_11_9.firstName._lEqual (firstName));
    }

    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () == Person.class &&
            _lEqual ((Person) _lArg));
    }
```

```
}


// End of file.
```

## E.15 PriorityQueue.java

```
package oolibrary;


import Ertsys.*;

// Packages imported
import oolibrary.*;


class _n1_PriorityQueue extends _eAny
{
    protected _n1_Heap myQueue;
    public _n1_PriorityQueue ()
    {
        super ();
        myQueue = new _n1_Heap ();
    }

    public void insert (_eAny a)
    {
        _n1_Heap _vUnshare_17_14 = ((_n1_Heap) myQueue._lClone ());
        myQueue = _vUnshare_17_14;
        _vUnshare_17_14.insert (a);
    }

    public _eAny getElement ()
    {
        if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
            )
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                if (!((!myQueue.empty ()))) throw new _xPre ("PriorityQueue.pd:20,13");
            }
            catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
        return myQueue.largest ();
    }

    public void remove ()
    {
        if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
            )
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                if (!((!myQueue.empty ()))) throw new _xPre ("PriorityQueue.pd:24,14");
            }
```

```
            catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
        _n1_PriorityQueue _vAntiAlias_self_25_22 = this;
        _n1_Heap _vUnshare_25_14 = ((_n1_Heap) myQueue._lClone ());
        myQueue = _vUnshare_25_14;
        _vUnshare_25_14.remove (_vAntiAlias_self_25_22.getElement ());
    }

    public _eSet ran ()
    {
        return myQueue.ran ();
    }

    public boolean _lEqual (_n1_PriorityQueue _vArg_11_9)
    {
        if (this == _vArg_11_9) return true;
        return _vArg_11_9.myQueue._lEqual (myQueue);
    }

    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () == _n1_PriorityQueue.
            class && _lEqual ((_n1_PriorityQueue) _lArg));
    }

}

class _n1_Heap extends _eAny
{
    final void _lc_Heap (String _lArg)
    {
        if (_eSystem.enableClassInvariantItem && _eSystem.currentCheckNesting <= _eSystem.
            maxCheckNesting)
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                if (!(_n1_Heap.isHeap (l, (_eTemplate_0) null))) throw new
                    _xClassInvariantItem ("PriorityQueue.pd:43,15", _lArg);
            }
            catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
    }

    void _lClassInvariantCheck (String _lArg)
    {
        _lc_Heap (_lArg);
    }

    protected _eSeq l;
    protected static _eSeq heapify (_eSeq xs, _eTemplate_0 _t0xs, int low, int high)
    {
        if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
            )
        {
```

```
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!((!xs.empty ()))) throw new _xPre ("PriorityQueue.pd:48,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!((((low <= high) && (high <= xs._oHash ())) && (0 < low)))) throw new
                _xPre ("PriorityQueue.pd:49,14");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    int _vLet_large_51_17 = (2 * low);
    if (_eSystem.enableEmbeddedAssert && _eSystem.currentCheckNesting <= _eSystem.
        maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!((1 < _vLet_large_51_17))) throw new _xEmbeddedAssert (
                "PriorityQueue.pd:51,43");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    return ((_vLet_large_51_17 <= high) ?
    ((((_vLet_large_51_17 < high) && (xs._oIndex (_eSystem._oPred (_vLet_large_51_17))
        .priority () < xs._oIndex (_vLet_large_51_17).priority ())) && (xs._oIndex (
        _eSystem._oPred (low)).priority () < xs._oIndex (_vLet_large_51_17).priority (
        ))) ?
    _n1_Heap.heapify (_n1_Heap.exchange (xs, (_eTemplate_0) null, low, (1 +
        _vLet_large_51_17)), (_eTemplate_0) null, (1 + _vLet_large_51_17), high) :
    (xs._oIndex (_eSystem._oPred (low)).priority () < xs._oIndex (_eSystem._oPred (
        _vLet_large_51_17)).priority ()) ?
    _n1_Heap.heapify (_n1_Heap.exchange (xs, (_eTemplate_0) null, low,
        _vLet_large_51_17), (_eTemplate_0) null, _vLet_large_51_17, high) :
    xs) :
    xs);
}

protected _eSeq sift_up (_eSeq xs, _eTemplate_0 _t0xs, int pos)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
```

```
        {
            if (!((0 < pos) && (pos <= xs._oHash ()))) throw new _xPre (
                "PriorityQueue.pd:67,14");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    int _vLet_parentPos_69_17 = _eSystem._oDiv (pos, 2);
    return ((1 == pos) ?
    xs :
    (xs._oIndex (_eSystem._oPred (pos)).priority () <= xs._oIndex (_eSystem._oPred (
        _vLet_parentPos_69_17)).priority ()) ?
    xs :
    sift_up (_n1_Heap.exchange (xs, (_eTemplate_0) null, _vLet_parentPos_69_17, pos),
        (_eTemplate_0) null, _vLet_parentPos_69_17));
}

protected _eSeq sift_down (_eSeq xs, _eTemplate_0 _t0xs)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!((!xs.empty ()))) throw new _xPre ("PriorityQueue.pd:81,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    return ((xs._oHash () == 1) ?
    new _eSeq () :
    _n1_Heap.heapify (_n1_Heap.exchange (xs, (_eTemplate_0) null, 1, xs._oHash ()).
        front (), (_eTemplate_0) null, 1, _eSystem._oPred (xs._oHash ())));
}

public _n1_Heap ()
{
    super ();
    l = new _eSeq ();
    _lc_Heap ("PriorityQueue.pd:90,14");
}

public static boolean isHeap (_eSeq a, _eTemplate_0 _t0a)
{
    boolean _vQuantifierResult_93_12;
    {
        _vQuantifierResult_93_12 = true;
        int _vCaptureCount_p_93_25 = _eSystem._oDiv (a._oHash (), 2);
        int _vLoopCounter_93_19 = 1;
        for (;;)
        {
            if (((_vCaptureCount_p_93_25 < _vLoopCounter_93_19) || (!
                _vQuantifierResult_93_12))) break;
            _vQuantifierResult_93_12 = _n1_Heap.greaterPriority (_vLoopCounter_93_19,
                a, (_eTemplate_0) null);
            if ((!_vQuantifierResult_93_12))
```

```
            {
            }
            else
            {
                _vLoopCounter_93_19 = _eSystem._oSucc (_vLoopCounter_93_19);
            }
        }
    }
    return _vQuantifierResult_93_12;
}

public static boolean greaterPriority (int p, _eSeq a, _eTemplate_0 _t0a)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!((0 < p) && (p <= a._oHash ()))) throw new _xPre (
                "PriorityQueue.pd:96,14");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    int _vLet_child_97_17 = (2 * p);
    return ((_vLet_child_97_17 < a._oHash ()) ?
    ((a._oIndex (_eSystem._oPred (_vLet_child_97_17)).priority () <= a._oIndex (
        _eSystem._oPred (p)).priority ()) && (a._oIndex ((_eSystem._oPred (
        _vLet_child_97_17) + 1)).priority () <= a._oIndex (_eSystem._oPred (p)).
        priority ())) :
    (a._oHash () == _vLet_child_97_17) ?
    (a._oIndex (_eSystem._oPred (_vLet_child_97_17)).priority () <= a._oIndex (
        _eSystem._oPred (p)).priority ()) :
    true);
}

public static _eSeq exchange (_eSeq xs, _eTemplate_0 _t0xs, int i, int j)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!((((i < j) && (j <= xs._oHash ())) && (0 < i)))) throw new _xPre (
                "PriorityQueue.pd:109,14");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    return xs.take (_eSystem._oPred (i)).append (xs._oIndex (_eSystem._oPred (j))).
        _oPlusPlus (xs.take (_eSystem._oPred (j)).drop (i).append (xs._oIndex (
        _eSystem._oPred (i))), (_eTemplate_0) null)._oPlusPlus (xs.take (xs._oHash ())
        .drop (j), (_eTemplate_0) null);
}
```

```
public static void swap (_eWrapper__eAny xs, _eSeq _t0xs, _eTemplate_0 _t1xs, int i,
    int j)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!((((i < j) && (j <= ((_eSeq) xs.value)._oHash ())) && (0 < i))))
                throw new _xPre ("PriorityQueue.pd:116,14");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    _eAny _vLet_temp_117_19 = ((_eSeq) xs.value)._oIndex (_eSystem._oPred (i));
    ((_eSeq) xs.value)._oaIndex (_eSystem._oPred (i), ((_eSeq) xs.value)._oIndex (
        _eSystem._oPred (j)));
    ((_eSeq) xs.value)._oaIndex (_eSystem._oPred (j), _vLet_temp_117_19);
}

public void insert (_eAny a)
{
    l = sift_up (l.append (a), (_eTemplate_0) null, (l._oHash () + 1));
    _lClassInvariantCheck ("PriorityQueue.pd:124,14");
}

public void remove (_eAny a)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!((!empty ()))) throw new _xPre ("PriorityQueue.pd:129,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(_eSystem._lEqual (a, largest ()))) throw new _xPre (
                "PriorityQueue.pd:130,20");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    l = sift_down (l, (_eTemplate_0) null);
    _lClassInvariantCheck ("PriorityQueue.pd:131,14");
}
```

```
    public _eAny largest ()
    {
        if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
            )
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                if (!((!empty ()))) throw new _xPre ("PriorityQueue.pd:136,13");
            }
            catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
        return l.head ();
    }

    public boolean empty ()
    {
        return l.empty ();
    }

    public _eSeq _rtoString ()
    {
        return l._rtoString ();
    }

    public _eSet ran ()
    {
        return l.ran ();
    }

    public boolean _ovIn (_eAny a)
    {
        return l._ovIn (a);
    }

    public boolean _lEqual (_n1_Heap _vArg_41_9)
    {
        if (this == _vArg_41_9) return true;
        return _vArg_41_9.l._lEqual (l);
    }

    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () == _n1_Heap.class &&
            _lEqual ((_n1_Heap) _lArg));
    }

}


// End of file.
```

# E.16 Staff.java

```
package oolibrary;
```

```
import Ertsys.*;

// Packages imported
import oolibrary.*;


public class Staff extends Person
{
    protected int staffId;
    protected int salary;
    public Staff (_eSeq first, char _t0first, _eSeq last, char _t0last, _eSeq dOB, char
        _t0dOB, int _vstaffId, int _vsalary)
    {
        super (first, (char) 0, last, (char) 0, dOB, (char) 0);
        staffId = _vstaffId;
        salary = _vsalary;
    }

    public _eSeq _rtoString ()
    {
        return _eSystem._ltoString (staffId)._oPlusPlus (_eSystem._lString (":"), (
            _eTemplate_0) null)._oPlusPlus (getName (), (_eTemplate_0) null);
    }

    public int getId ()
    {
        return staffId;
    }

    public void increaseSalary (int inc)
    {
        salary = (salary + inc);
    }

    public boolean _lEqual (Staff _vArg_12_9)
    {
        if (this == _vArg_12_9) return true;
        if (!super._lEqual (_vArg_12_9)) return false;
        return ((_vArg_12_9.staffId == staffId) && (_vArg_12_9.salary == salary));
    }

    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () == Staff.class &&
            _lEqual ((Staff) _lArg));
    }

}


// End of file.
```

# E.17 StaffBase.java

```
public class StaffBase extends _eAny
{
    final void _lc_StaffBase (String _lArg)
    {
```

```
    if (_eSystem.enableClassInvariantItem && _eSystem.currentCheckNesting <= _eSystem.
        maxCheckNesting)
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            boolean _vQuantifierResult_13_15;
            {
                _vQuantifierResult_13_15 = true;
                int _vCaptureCount_s1_13_29 = allStaff._oHash ();
                int _vLoopCounter_13_22 = 0;
                for (;;)
                {
                    if (((_vLoopCounter_13_22 == _vCaptureCount_s1_13_29) || (!
                        _vQuantifierResult_13_15))) break;
                    boolean _vQuantifierResult_13_25;
                    {
                        _vQuantifierResult_13_25 = true;
                        int _vCaptureCount_s2_13_29 = allStaff._oHash ();
                        int _vLoopCounter_13_25 = 0;
                        for (;;)
                        {
                            if (((_vLoopCounter_13_25 == _vCaptureCount_s2_13_29) || (
                                !_vQuantifierResult_13_25))) break;
                            _vQuantifierResult_13_25 = (!(!((Staff) allStaff._oIndex (
                                _vLoopCounter_13_22))._lEqual (((Staff) allStaff.
                                _oIndex (_vLoopCounter_13_25)))) || (!((Staff)
                                allStaff._oIndex (_vLoopCounter_13_22)).getId () == ((
                                Staff) allStaff._oIndex (_vLoopCounter_13_25)).getId (
                                ))));
                            if ((!_vQuantifierResult_13_25))
                            {
                            }
                            else
                            {
                                _vLoopCounter_13_25 = _eSystem._oSucc (
                                    _vLoopCounter_13_25);
                            }
                        }
                    }
                    _vQuantifierResult_13_15 = _vQuantifierResult_13_25;
                    if ((!_vQuantifierResult_13_15))
                    {
                    }
                    else
                    {
                        _vLoopCounter_13_22 = _eSystem._oSucc (_vLoopCounter_13_22);
                    }
                }
            }
            if (!(_vQuantifierResult_13_15)) throw new _xClassInvariantItem (
                "StaffBase.pd:13,15", _lArg);
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
}

void _lClassInvariantCheck (String _lArg)
```

```
{
    _lc_StaffBase (_lArg);
}

protected _eSet allStaff;
public StaffBase ()
{
    super ();
    allStaff = new _eSet ();
    _lc_StaffBase ("StaffBase.pd:16,14");
}

public boolean uniqueId (Staff s)
{
    return (!usedIds ()._ovIn (((_eAny) new _eWrapper_int (s.getId ()))));
}

public _eSet usedIds ()
{
    _eSet _vForYield_22_12;
    {
        _vForYield_22_12 = new _eSet ();
        int _vCaptureCount_s_22_19 = allStaff._oHash ();
        int _vLoopCounter_22_16 = 0;
        for (;;)
        {
            if ((_vLoopCounter_22_16 == _vCaptureCount_s_22_19)) break;
            _vForYield_22_12 = _vForYield_22_12.append (((_eAny) new _eWrapper_int (((
                Staff) allStaff._oIndex (_vLoopCounter_22_16)).getId ())));
            _vLoopCounter_22_16 = _eSystem._oSucc (_vLoopCounter_22_16);
        }
    }
    return _vForYield_22_12;
}

public void addStaff (Staff sta)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
        try
        {
            if (!(uniqueId (sta))) throw new _xPre ("StaffBase.pd:25,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    allStaff = allStaff.append (((_eAny) sta));
    _lClassInvariantCheck ("StaffBase.pd:26,14");
}

public boolean _lEqual (StaffBase _vArg_11_9)
{
    if (this == _vArg_11_9) return true;
    return _vArg_11_9.allStaff._lEqual (allStaff);
}

public boolean equals (_eAny _lArg)
```

```
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () == StaffBase.class &&
            _lEqual ((StaffBase) _lArg));
    }

}


// End of file.
```

## E.18 UserBase.java

```
package oolibrary;


import Ertsys.*;

// Packages imported
import oolibrary.*;


public class UserBase extends _eAny
{
    protected BorrowerBase allBorrowers;
    protected StaffBase allStaff;
    public UserBase ()
    {
        super ();
        allBorrowers = new BorrowerBase ();
        allStaff = new StaffBase ();
    }

    public _eSet usedIds ()
    {
        return allBorrowers.usedIds ()._oPlusPlus (allStaff.usedIds (), (_eTemplate_0)
            null);
    }

    public boolean uniqueId (Borrower u)
    {
        return (!hasId (u.getId ()));
    }

    public boolean uniqueId (Staff u)
    {
        return (!hasId (u.getId ()));
    }

    public boolean hasId (int uId)
    {
        return usedIds ()._ovIn (((_eAny) new _eWrapper_int (uId)));
    }

    public Borrower findUser (int id)
    {
        if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
            )
        {
            _eSystem.currentCheckNesting ++;
            try
```

```
        {
            if (!(hasId (id))) throw new _xPre ("UserBase.pd:33,13");
        }
        catch (_xCannotEvaluate _lException)
        {
        }
        _eSystem.currentCheckNesting --;
    }
    Borrower _vChoose_34_12 = null;
    {
        _eSet _vCaptureBound_user_34_36 = allBorrowers.allTheBorrowers ();
        boolean _vSelectorCondition_34_12;
        _vSelectorCondition_34_12 = false;
        int _vCaptureCount_user_34_36 = _vCaptureBound_user_34_36._oHash ();
        int _vLoopCounter_34_17 = 0;
        for (;;)
        {
            if (((_vLoopCounter_34_17 == _vCaptureCount_user_34_36) ||
                _vSelectorCondition_34_12)) break;
            _vSelectorCondition_34_12 = (((Borrower) _vCaptureBound_user_34_36._oIndex
                (_vLoopCounter_34_17)).getId () == id);
            if (_vSelectorCondition_34_12)
            {
                _vChoose_34_12 = ((Borrower) _vCaptureBound_user_34_36._oIndex (
                    _vLoopCounter_34_17));
            }
            else
            {
            }
            if (_vSelectorCondition_34_12)
            {
            }
            else
            {
                _vLoopCounter_34_17 = _eSystem._oSucc (_vLoopCounter_34_17);
            }
        }
        if (_eSystem.enableThatOrAny && _eSystem.currentCheckNesting <= _eSystem.
            maxCheckNesting)
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                if (!(_vSelectorCondition_34_12)) throw new _xThatOrAny (
                    "UserBase.pd:34,12");
            }
            catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
    }
    return _vChoose_34_12;
}

public void addBorrower (Borrower bor)
{
    if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
        )
    {
        _eSystem.currentCheckNesting ++;
```

```
            try
            {
                if (!(uniqueId (bor))) throw new _xPre ("UserBase.pd:37,13");
            }
            catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
        BorrowerBase _vUnshare_38_14 = ((BorrowerBase) allBorrowers._lClone ());
        allBorrowers = _vUnshare_38_14;
        _vUnshare_38_14.addBorrower (bor);
    }

    public void addStaff (Staff sta)
    {
        if (_eSystem.enablePre && _eSystem.currentCheckNesting <= _eSystem.maxCheckNesting
            )
        {
            _eSystem.currentCheckNesting ++;
            try
            {
                if (!(uniqueId (sta))) throw new _xPre ("UserBase.pd:41,13");
            }
            catch (_xCannotEvaluate _lException)
            {
            }
            _eSystem.currentCheckNesting --;
        }
        StaffBase _vUnshare_42_14 = ((StaffBase) allStaff._lClone ());
        allStaff = _vUnshare_42_14;
        _vUnshare_42_14.addStaff (sta);
    }

    public boolean _lEqual (UserBase _vArg_11_9)
    {
        if (this == _vArg_11_9) return true;
        return (_vArg_11_9.allBorrowers._lEqual (allBorrowers) && _vArg_11_9.allStaff.
            _lEqual (allStaff));
    }

    public boolean equals (_eAny _lArg)
    {
        return _lArg == this || (_lArg != null && _lArg.getClass () == UserBase.class &&
            _lEqual ((UserBase) _lArg));
    }

}


// End of file.
```

# Appendix F

# Resource Manager Specification Code

Here is the complete specification of the Resource Manager.

## F.1 ProcessErrorCode.pd

```
class ProcessErrorCode ^=
    enum
        running,        // the process is executing
        sleeping,       // the process is waiting on a resource
        terminated      // the process has terminated successfully
    end;
```

## F.2 ProcessItem.pd

```
class ProcessItem ^=
inherits ResourceItem
abstract
    var
        state:ProcessStateCode,
        hasResources:set of from ResourceItem,
        needsResources:set of from ResourceItem;

        invariant hasResources**needsResources=set of from ResourceItem{};
interface

//  constructor
    build{i:nat,!state:ProcessStateCode,
                            !hasResources:set of from ResourceItem,
                !needsResources:set of from ResourceItem}
```

```
        pre hasResources**needsResources= set of from ResourceItem{}
        inherits ResourceItem{i};

//  constructor that creates a process with no needs.
    build{i:nat}
        ^= ProcessItem{i,running@ProcessStateCode,
                       set of from ResourceItem{},
                       set of from ResourceItem{}};

//  constructor that creates a process with needs.
    build{i:nat,needs:set of from ResourceItem}
        ^= ProcessItem{i,running@ProcessStateCode,
                       set of from ResourceItem{},
                       needs};
//**
//** Methods controlling & describing state of a processItem
//**
//  determines if this process has met all it's needs
    function allNeedsMet:bool
        ^= #needsResources=0;

//  determines if this process has terminated
    function terminated:bool
        ^= state=terminated@ProcessStateCode;

//  determines if this process has running
    function running:bool
        ^= state=running@ProcessStateCode;

//  determines if the process is currently sleeping
    function sleeping:bool
        ^= state=sleeping@ProcessStateCode;

//  determines if this process is alive
    function alive:bool
        ^= ~terminated;

//  puts a process to sleep(symbolically),
//  it must not be sleeping already
    schema !sleep
        pre ~sleeping
        post state!=sleeping@ProcessStateCode;

//  terminate the process, given that it isn't holding onto
//  any resources and has met all it's needs
//          => held all the resources it needed originally
    schema !terminate
        pre ~holdingResources,
            allNeedsMet
        post state!=terminated@ProcessStateCode;

//  force a termination on a process regardless of needs met
    schema !forceTerminate
        post state!=terminated@ProcessStateCode;

//**
//** Methods concerned with resource acquisition
//** and process execution
//**
//  determine if this process has needs
    function hasNeeds:bool
```

```
        ^=~needsResources.empty;

//  determines if this process needs a given resource
    function needs(r:from ResourceItem):bool
        ^= r in needsResources;

//  determines if this process currently holds any resource
    function holdingResources:bool
        ^= #hasResources ~= 0;

//  determines if this process currently holds a resource
    function holdingResource(r:from ResourceItem):bool
        ^= r in hasResources;

//  determines the Resources that the process holds
    redefine function next:set of from ResourceItem
        ^= hasResources;

//  pick a random need of the process : Random???
    function getNeed:from ResourceItem
        pre hasNeeds
        satisfy result in needsResources
        via
            value needsResources.min
        end;

//  pick a random resource the process has : Random???
    function getAcquired:from ResourceItem
        pre holdingResources
        satisfy result in hasResources
        via
            value hasResources.min
        end;

//  allows us to add needs to process item
    schema !addNeeds(needs:set of from ResourceItem)
        pre needs ** hasResources = set of from ResourceItem{}
        post needsResources!=needsResources++needs;

//  allocate a resource to this process item
    schema !granted(r:from ResourceItem)
        pre  needs(r), sleeping
        post needsResources!=needsResources.remove(r),
             hasResources!=hasResources.append(r),
             state!=running@ProcessStateCode;

//  called when the process item no longer needs a resource
    schema !remove(r:from ResourceItem)
        post hasResources!=hasResources.remove(r);

    function progress:nat
        ^= for r::needsResources yield r.processPosition(self);

end;
```

# F.3 ProcessStateCode.pd

```
class ProcessStateCode ^=
    enum
        running,        // the process is executing
```

```
        sleeping,       // the process is waiting on a resource
        terminated      // the process has terminated
    end;
```

## F.4 ResourceItem.pd

```
class ResourceItem ^=
abstract
    var
        id:nat,
        lock:Semaphore;

        invariant forall p::lock.waitingOn :- p.sleeping;
//        invariant forall p::lock.waitingOn :- p.needs(self);

interface
// This is required to use the "from ResourceItem" for object
// instantiation. This will allow equals to be defined on all
// objects in the inheritance hierarchy of ResourceItem
    operator = (arg);

//  constructor.
    build{!id:nat}
        post lock!=Semaphore{};

//  get the Id of the resource
    function getId:nat
        ^= id;

//  determine if the resource is available(i.e. lock is free)
    function isAvailable:bool
        ^= lock.isFree;

//  determine if the resource has processes waiting on it.
    function hasWaiting:bool
        ^= lock.hasWaiting;

//  get their next lot of resourceItems the system is waiting on.
    function next:set of from ResourceItem
        ^= for p::lock.waitingOn yield p as from ResourceItem;

//  declare a want of the resource
    schema !acquire(p!:ProcessItem)
        pre p.needs(self),~p.sleeping
        post lock!P1(p!);

//  actually retrive the resource
    schema !grant(p!:out ProcessItem)
        pre hasWaiting,isAvailable
        post lock!P2(p!)
        assert p'.sleeping;//, p'.needs(self);

    schema !free
        post ([lock.isFull]:
                // a process Item is freeing the resource
                // while the resource is not allocated
                // !!!!Indicates an error!!!!!
                pass,
              []:
               lock!V
```

```
            );

//  A reset schema, frees the resource from all responsibilities
    schema !reset
        post lock!reset;

    function processPosition(p:ProcessItem):nat
        ^= lock.lockPosition(p);

end;
```

# F.5 ResourceManager.pd

```
class ResourceManager ^=
abstract

    var
        id:nat;
//  The manager is the algorithm by which the resource allocation
//  scheme is implemented. By all rights it should be just a part
//  of the System, however we represent it as an object in order
//  that should the resource allocation scheme be changed, we can
//  use inheritance to re-use the resource manager code, and
//  change the the allocation scheme. The system will still be
//  represented by the no deadlock, no starvation, fairness
//  invariants but the resource manager will be the code that
//  implements them.

interface
    build{!id:nat};

//  A process states want of a resource, will be put to sleep
//  and on a subsequent grant of the resource, will be acquired
//  two stage operation owing to lack of multi-threaded support.
//  This method needs to terminate.
    schema acquire(p!:ProcessItem, r!:from ResourceItem)
        pre ~p.sleeping, p.needs(r)
        post r!acquire(p!);


//  The granting of a resource to an instantiated process. The
//  input parameter of p could be anything, but the return value
//  will be important. We need to signal if the resources can't
//  be granted owing to lack of availablity.
    schema grant(r!:from ResourceItem)
        pre r.hasWaiting
        post (var p:ProcessItem;
              [r.isAvailable]:
                  r!grant(p!)then
                  p!granted(r),
              []:
                // Can't grant a process. Do we need to signal?
                pass
             );

//  Release a resource
    schema release(p!:ProcessItem, r!:from ResourceItem)
        pre p.holdingResource(r)
        post p!remove(r), r!free;
```

```
end;
```

## F.6 Semaphore.pd

```
class Semaphore ^=
abstract
    var
        maxSem:nat,     // maximum Value of the Semaphore
        currentSem:nat,  // value of semaphore
        waitingQ:seq of ProcessItem;    //items waiting on it

        invariant currentSem<=maxSem;
        invariant forall p::waitingQ :- p.sleeping;

interface

    build{}
        ^= Semaphore{1};
    build{!maxSem:nat}
        pre maxSem>0
        post currentSem!=maxSem,
             waitingQ!=seq of ProcessItem{};

//  test whether the Semaphore is Free or has been locked
    function isFree:bool
        ^= currentSem~=0;

    function hasWaiting:bool
        ^= ~waitingQ.empty;

//  test if the semaphore has it's maximum value to ensure
// the semaphore isn't released when it wasn't acquired
    function isFull:bool
        ^= currentSem=maxSem;

//  test on the number of processes waiting on the semaphore
    function count:nat
        ^= #waitingQ;

//  the method P needed to be segmented into two parts.
//  the first is always acceptable and deals with a process
//  wanting to be granted access to the semaphore
    schema !P1(item!:ProcessItem)
        pre ~item.sleeping
        post item!sleep then
             waitingQ!=waitingQ.append(item);

//  the second will only be accepted if the semaphore is free
//  and returns the process that now has the semaphore to allow
//  for it to be executed
    schema !P2(item!:out ProcessItem)
        pre isFree, ~waitingQ.empty
        post currentSem!=currentSem-1,
             item!=waitingQ.head,
             waitingQ!=waitingQ.tail;

//  the v method unlocks the semaphore, on a subsequent call P2
//  the semaphore will be allocated to the next item in the queue
    schema !V
        pre ~isFull
```

```
        post currentSem!=currentSem+1;

//  reset the semaphore object release it from responsibilities
    schema !reset
        post currentSem!=maxSem, waitingQ!=seq of ProcessItem{};

    function waitingOn:set of ProcessItem
        ^= waitingQ.ran;

    function lockPosition(p:ProcessItem):nat
        ^= ([p in waitingQ]:
                waitingQ.findFirst(p),
            []:
                #waitingQ
           );
end;
```

# F.7 System.pd

```
// System with one resource, no processes is not in deadlock
property assert ~System{
                    set of from ResourceItem{ResourceItem{1}},
                    set of ProcessItem{}}.hasDeadlock;

// System with one resource, one processes is not in deadlock
property assert ~(System{
                  set of from ResourceItem{ResourceItem{1}, ProcessItem{2}},
                  set of ProcessItem{ProcessItem{2}}} after it!step(1)).hasDeadlock;

class System ^=
abstract
    var
        resources:set of from ResourceItem,
        manager:ResourceManager,
        toBeExecuted:set of ProcessItem;

    invariant forall r::resources :- r.next<<=resources;

//  deadlock detection algorithm
//  must be placed before the invariant that declares
//  the system doesn't contain deadlock
//    function hasDeadlock:bool
//        ^=([resources.empty]:
//               false,
//           []:
//               exists r::resources :- hasCycle(set of from ResourceItem{},r,r.next.rep(1))
//          );

    function hasDeadlock:bool
        ^=self after it!step.progress > self.progress;

    function hasCycle(viewed:set of from ResourceItem,at:from ResourceItem,
                      toBeViewed:bag of from ResourceItem):bool
        pre viewed <<= resources,
            at in resources,
            toBeViewed.ran <<= resources
        decrease #resources - #viewed
        ^= ([toBeViewed.empty]:
                false,
            [at in viewed]:
```

```
                true,
             []: (let newAt^=toBeViewed.min;
                  hasCycle(viewed.append(at),newAt,toBeViewed.remove(newAt)++newAt.next.rep(1))
                 )
            );

        invariant ~hasDeadlock;

interface

    nonmember function processesAreResources(res:set of from ResourceItem,
                               pro:set of ProcessItem):bool
    ^= (let processesAsResources ^=
                        for p::pro yield p as from ResourceItem;
        processesAsResources <<= res
       );

//  constructors
    build{res:set of from ResourceItem,
          man:ResourceManager,
          toBeExe:set of ProcessItem}
        pre processesAreResources(res,toBeExe),
            forall r::res :- r.next=set of from ResourceItem{}
        post resources!=res,
             manager!=man,
             toBeExecuted!=toBeExe;

    build{res:set of from ResourceItem,
          toBeExe:set of ProcessItem}
        pre processesAreResources(res,toBeExe),
            forall r::res :- r.next=set of from ResourceItem{}

        post resources!=res,
             manager!=ResourceManager{1},
             toBeExecuted!=toBeExe;
//  empty system
    build{}^= System{set of from ResourceItem{},ResourceManager{1},set of ProcessItem{}};

//  determines that all the processes have terminated
//  (i.e. system has completed)
    function systemEnd:bool
        ^= forall p::toBeExecuted :-p.terminated;

//  collections of processes of particular state in our system
    function deadProcesses:set of ProcessItem
        ^= those p::toBeExecuted :- p.terminated;

    function activeProcesses:set of ProcessItem
        ^= those p::toBeExecuted :- p.running;

    function hungryProcesses:set of ProcessItem
        ^= those p::toBeExecuted :- p.running & p.hasNeeds;

    function hasHungry:bool
        ^= ~hungryProcesses.empty;

    function sleepingProcesses:set of ProcessItem
        ^= those p::toBeExecuted :- p.sleeping;

    function hasSleeping:bool
        ^= ~sleepingProcesses.empty;
```

```
    function holdingResources:set of ProcessItem
        ^= those p::toBeExecuted :- p.holdingResources;

    function hasAcquired:bool
        ^= ~holdingResources.empty;

    function neededResources:set of from ResourceItem
        ^= those r::resources :- r.hasWaiting;

    function hasNeeded:bool
        ^= ~neededResources.empty;

//  pick a random active process : Random???
    function pickActive:ProcessItem
        pre ~activeProcesses.empty
        satisfy result in activeProcesses
        // current implementation just picks the minimum element
        // there should be some randomisation of this...?
        via value activeProcesses.min
        end;

//  pick a random hungry process : Random???
    function pickHungry:ProcessItem
        pre hasHungry
        satisfy result in hungryProcesses
        // current implementation just picks the minimum element
        // there should be some randomisation of this...?
        via value hungryProcesses.min
        end;

//  pick a random sleeping process : Random???
    function pickSleeping:ProcessItem
        pre hasSleeping
        satisfy result in sleepingProcesses
        // current implementation just picks the minimum element
        // there should be some randomisation of this...?
        via value sleepingProcesses.min
        end;

//  pick a random process that holds a resource : Random???
    function pickHasAcquired:ProcessItem
        pre hasAcquired
        satisfy result in holdingResources
        // current implementation just picks the minimum element
        // there should be some randomisation of this...?
        via value holdingResources.min
        end;

//  pick a random needed resource : Random???
    function pickNeededResource:from ResourceItem
        pre hasNeeded
        satisfy result in neededResources
        // current implementation just picks the minimum element
        // there should be some randomisation of this...?
        via value neededResources.min
        end;

    function hasDeadlock;

    function progress:nat
```

```
        ^= for p::toBeExecuted yield p.progress;

//  resource manager steps the system.
    schema !step(choice:nat)
        pre ~systemEnd      // there is work to do.
        post (var p:ProcessItem,r:from ResourceItem;
              [choice=0]:   // declare want of resource
                ([hasHungry]:
                    p!=pickHungry then
                    resources!=resources.remove(p) then
                    toBeExecuted!=toBeExecuted.remove(p) then
                    r!=p.getNeed then
                    resources!=resources.remove(r) then
                    manager.acquire(p!,r!) then
                    resources!=resources.append(p) then
                    toBeExecuted!=toBeExecuted.append(p) then
                    resources!=resources.append(r),
                 []:
                    pass    // System has allocated all it needs
                            // we don't need to acquire any more
                ),
              [choice=1]:   // grant the resource to a process
                ([hasNeeded]:
                    r!=pickNeededResource then
                    resources!=resources.remove(r) then
                    manager.grant(r!) then
                    resources!=resources.append(r),
                 []:
                    pass    // System has no processes waiting on
                            // resources currently.
                ),
              []:           // release a resource
                ([hasAcquired]:
                    p!=pickHasAcquired then
                    resources!=resources.remove(p) then
                    toBeExecuted!=toBeExecuted.remove(p) then
                    r!=p.getAcquired then
                    resources!=resources.remove(r) then
                    manager.release(p!,r!) then
                    ([~p.holdingResources & p.allNeedsMet]:
                        p!terminate,    // end of life
                     []:
                        pass
                    )then
                    resources!=resources.append(p) then
                    toBeExecuted!=toBeExecuted.append(p) then
                    resources!=resources.append(r),

                 []:
                    pass    // the system currently has no
                                  // resources allocated currently
                )
                //  should the 'pass' routes of execution
                //  signal errors?
              );


end;
```

# Appendix G

# Exception Handling

Here is the Perfect specification and Java front end of an application that should cause an exception in software, but doens't, yet continues to execute after entering an inconsisten state.

## G.1 Examples.pd

```
//*****************************************************
//* File:    H:\PerfectTutorial\Examples.pd
//* Author:  Automatically generated by Perfect Developer
//* Created: 09:47:44 on Monday December 8th 2003 UTC
//*****************************************************

property assert 2+2=4;

// Excercise 1. Q1

const firstHundredInts :seq of int
    ^= 0..100;

property assert 42 in firstHundredInts;

property assert 101 ~in firstHundredInts;

// Q2

function division (i,j:int) : bool
    pre j>0
    ^= i%j=0;

//Q3

const prime2to100 : seq of int
```

```
    ^= those x::2..100 :- isPrime(x);

function isPrime(x:int) : bool
    ^= forall i::2..<x :- x%i ~=0;

property assert prime2to100.isndec;

property assert 53 in prime2to100;

property assert 18 ~in prime2to100;

// Q4

function max2Sets(set1, set2 : seq of char) : char
    pre ~set1.empty, ~set2.empty
    ^= max(set1.max,set2.max);

//property assert max2Sets(set of int{1,2}, set of int{3}) = 3;

// Q5

const squaresPrime : seq of int
    ^= for x::prime2to100 yield x*x;

property assert 53*53 in squaresPrime;

property assert 18*18 ~in squaresPrime;
```

# G.2 Application.pd

```
//*****************************************************
//* File:    C:\Work\Escher\Examples\1-day tutorial\Application.pd
//* Author:  DC
//* Created: 17:27:22 on Sunday November 23rd 2003 UTC
//*****************************************************

import "Examples.pd";

// This is a skeleton application backend for you to start from
class Application ^=
abstract
    var rslt: string,
        tot: int;

    // Function to test whether a parameter is a digit string
    nonmember function isNumber(s: string): bool
        ^= ~s.empty & (forall x::s :- x.isDigit);

    // Add two numbers and return the result as a string
    nonmember function add(a, b: int): string
        ^= (a + b).toString;

    // Add one number to the total and subtract the other from it
    // Copy the new total to the result string
    schema !sum(a, b: int)
        post tot! = tot + a - b,
            rslt! = tot'.toString;

interface
    // These interface methods and constructor are called directly from the Java front-end.
```

```
    // Therefore, they should have no preconditions, nor any parameters of constrained types.

    // This is is called when you press the "Call function1" button
    // It returns the result of concatenating the parameters
    function function1(p1, p2: string): string
        ^= p1 ++ p2;

    // This is is called when you press the "Call function2" button
    // If both parameters are numbers, it returns the result of calling function 'add'
    function function2(p1, p2: string): string
        ^=  ( [isNumber(p1) & isNumber(p2)]:
                  add(int{p1}, int{p2}),
              []:
                  "Invalid parameter"
            );

    // This is is called when you press the "Call schema1" button
    // It clears the total and returns "Total is clear"
    schema !schema1(p1, p2: string)
        post rslt! = "Total is clear", tot! = 0;

    // This is is called when you press the "Call schema2" button.
    // If both parameters are numbers, it converts them to integers and calls schema '!sum'
    schema !schema2(p1, p2: string)
        post (  [isNumber(p1) & isNumber(p2)]:
                    !sum(int{p1}, int{p2}),
                []:
                    rslt! = "Invalid parameter"
             );

    // This is called to get the result to display after calling schema or schema2
    function getResult: string
        ^= rslt;

    // This is the constructor called from the Java front-end to create the application backend object
    build{}
        post rslt! = "result not set", tot! = 0;
end;

// End
```

# G.3 Tutorial.java

```
// Simple Java Swing application to front-end a Perfect Developer program

package Tutorial;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import Ertsys.*;

// This is the top-level class
public class TutorialExamples implements ActionListener
{
    JFrame appFrame;                            // Swing object for the entire window
    JPanel appPanel;                            // Swing object for the panel
    JTextField textInput1, textInput2;          // Swing objects for the parameter input text boxes
    JLabel status;                              // Swing object for displaying the result
```

```
JButton button1, button2, button3, button4;   // Buttons that the user can press
    Application backend;                      // 'Perfect' backend

// Constructor
public TutorialExamples()
    {
        // Create the back end
        backend = new Application();

        // Create the frame and container.
        appFrame = new JFrame("Tutorial example");
        appPanel = new JPanel();
        appPanel.setLayout(new GridLayout(0, 2, 10, 10));
        appPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        // Add the widgets.
        addWidgets();

        // Add the panel to the frame.
        appFrame.getContentPane().add(appPanel, BorderLayout.CENTER);

        // Exit when the window is closed.
    appFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Show the app.
        appFrame.pack();
        appFrame.setVisible(true);
}

// Create and add the widgets for app.
private void addWidgets()
    {
        // Create widgets.
        textInput1 = new JTextField(20);
        textInput2 = new JTextField(20);
        status = new JLabel();
        button1 = new JButton("Call 'function1'");
        button2 = new JButton("Call 'function2'");
        button3 = new JButton("Call 'schema1' and get result");
        button4 = new JButton("Call 'schema2' and get result");

        // Listen to events from Convert button.
        button1.addActionListener(this);
        button2.addActionListener(this);
        button3.addActionListener(this);
        button4.addActionListener(this);

        // Add widgets to container.
        appPanel.add(new JLabel("Parameter 1:", JLabel.RIGHT));
        appPanel.add(textInput1);
        appPanel.add(new JLabel("Parameter 2:", JLabel.RIGHT));
        appPanel.add(textInput2);
        appPanel.add(button1);
        appPanel.add(button2);
        appPanel.add(button3);
        appPanel.add(button4);
        appPanel.add(new JLabel("Result:", JLabel.RIGHT));
        appPanel.add(status);
}

// Implementation of ActionListener interface.
```

```
    public void actionPerformed(ActionEvent event)
        {
            String param1 = textInput1.getText();
            String param2 = textInput2.getText();
            if (event.getSource() == button1)
            {
                status.setText(_eSystem._lJavaString(backend.function1
                (_eSystem._lString(textInput1.getText()), (char)0,
                _eSystem._lString(textInput2.getText()), (char)0)));
            }
            else if (event.getSource() == button2)
            {
                status.setText(_eSystem._lJavaString(backend.function2
                (_eSystem._lString(textInput1.getText()), (char)0,
                _eSystem._lString(textInput2.getText()), (char)0)));
            }
            else if (event.getSource() == button3)
            {
                backend.schema1(_eSystem._lString(textInput1.getText()),
                 (char)0, _eSystem._lString(textInput2.getText()), (char)0);
                status.setText(_eSystem._lJavaString(backend.getResult()));
            }
            else if (event.getSource() == button4)
            {
                backend.schema2(_eSystem._lString(textInput1.getText()),
                (char)0, _eSystem._lString(textInput2.getText()), (char)0);
                status.setText(_eSystem._lJavaString(backend.getResult()));
            }
            else status.setText("Unknown event or return code");
        }

    // main method
    public static void main(String[] args)
        {
            // Set the look and feel.
            try
            {
              UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
            } catch(Exception e) {}

            TutorialExamples applicationObject = new TutorialExamples();
    }
}

// End
```

# Bibliography


[1] Escher Technologies website: `http://www.eschertech.com/index.php`.

[2] Borland : Together Technologies website: `http://www.togethersoft.com/`.

[3] Atelier B website: `http://www.atelierb.societe.com/index_uk.htm`.

[4] Francois Taiani, Mario Paludetto, Thierry Cros, Avoiding state explosion: A brief introduction to binary branching diagrams and petri net unfoldings. Technical Report LAAS No00377, October 15 2002.

[5] *The Perfect Developer Language Reference Manual, Version 3.0 :* ***http://www.eschertech.com/product_documentation/LanguageReference/language_reference.pdf***, 2004.

[6] Martin Abadi and Luca Cardelli. *A Theory of Objects*. Monographs in Computer Science. Springer-Verlang, 1998.

[7] J.-R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[8] Wolfgang Ahrendt, Thomas Baar, Bernhard Beckert, Richard Bubel, Martin Giese, Reiner Hähnle, Wolfram Menzel, Wojciech Mostowski, Andreas Roth,

Steffen Schlager, and Peter H. Schmitt. The KeY tool. *Software and System Modeling*, 4:32–54, Springer, 2005.

[9] Sinan Si Alhir. *UML in a Nutshell*. O'Reilly, 1998.

[10] B-Core. The b-toolkit. http://www.b-core.com/btoolkit.html.

[11] B. Weisssman B. Gomes, D. Stoutamire and H. Klawitter. *Sather 1.1 : Language Essentials*, 1.1 edition, 1996. `http://www.icsi.berkeley.edu/~sather/Documentation/LanguageDescription/contents.html`.

[12] Ralph-Johan Back and Joakim von Wright. *Refinement Calculus: A Systematic Introduction*. Graduate Texts in Computer Science. Springer-Verlag, 1998.

[13] Michael Baentsch, Peter Buhler, Thomas Eirich, Frank Höring, and Marcus Oestreicher. JavaCard — from hype to reality. *IEEE Concurrency*, 7(4):36–43, October – December 1999.

[14] Thomas Ball, Byron Cook, Vladimir Levin, and Sriram K. Rajamani. SLAM and static driver verifier: Technology transfer of formal methods inside microsoft. Technical Report MSR-TR-2004-08, Microsoft Research (MSR), January 2004.

[15] Thomas Ball, Byron Cook, Vladimir Levin, and Sriram K. Rajamani. Slam and static driver verifier: Technology transfer of formal methods inside microsoft. In *Proc. IFM 2004*, pages 1–20, Springer, 2004.

[16] Peter Becker and D. Reilly. C++ templates. *Dr. Dobb's Journal of Software Tools*, 18(8):44, 46–51, 102–103, August 1993. `http://www.ddj.com/`.

[17] Bernhard Beckert, Martin Giese, Elmar Habermalz, Reiner Hähnle, Andreas Roth, Philipp Rümmer, and Steffen Schlager. Taclets: A New Paradigm for Constructing Interactive Theorem Provers. Fachberichte Informatik 9–2004, Universität Koblenz-Landau, Universität Koblenz-Landau, Institut für Informatik, Universitätsstr. 1, D-56070 Koblenz, 2004.

[18] P. Bellini, R. Mattolini, and P. Nesi. Temporal logics for real-time system specification. *ACM Computing Surveys*, 32(1):12–42, March 2000.

[19] Antoine Beugnard. OO languages late-binding signature. In *Informal Workshop Record of FOOL 9 (The Ninth International Workshop on Foundations of Object-Oriented Languages)*, pages 61–66, 2002.

[20] Wolfgang Bibel and Peter H. Schmidt, editors. *Automated Deduction: A Basis for Applications. Volume III, Applications*, volume 1. Kluwer Academic Publishers, Dordrecht, 1998.

[21] J. C. Bicarregui, C A R Hoare, and J C P Woodcock. The verified software repository: a step towards the verifying compiler, preprint, 2005.

[22] Richard J. Bird and Philip Wadler. *Introduction to Functional Programming*. International Series in Computer Science. Prentice-Hall, New York, NY, 1988.

[23] Kim Bruce. *Foundations of Object Oriented Programming Languages: Types and Semantics*. The MIT Press, 2002.

[24] Kim B. Bruce, Angela Schuett, and Robert van Gent. PolyTOIL: A type-safe polymorphic object-oriented language. *Lecture Notes in Computer Science*, 952:27–51, Springer, 1995.

[25] Luca Cardelli. Typeful programming. In E. J. Neuhold and M. Paul, editors, *Formal Description of Programming Concepts*, pages 431–507. Springer-Verlag, Berlin, 1991.

[26] Gareth Carter and Rosemary Monahan. Introducing the perfect language. Technical Report NUIM-CS-TR-2005-06, National Universtiy of Ireland Maynooth, Department of Computer Science, 2005.

[27] Gareth Carter and Rosemary Monahan. Software specification, implementation and execution with perfect. Technical Report NUIM-CS-TR-2005-07, National Universtiy of Ireland Maynooth, Department of Computer Science, 2005.

[28] Gareth Carter, Rosemary Monahan, and Joseph M. Morris. Software refinement with perfect developer. In *Proc. SEFM 2005*, IEEE, 2005.

[29] Giuseppe Castagna. Covariance and contravariance: Conflict without a cause. *ACM Transactions on Programming Languages and Systems*, 17(3):431–447, May 1995.

[30] ClearSy. B4free and click'n'prove. `http://www.b4free.com/index.php`.

[31] Compaq Computer Corporation. Extended static checking for Java. `http://research.compaq.com/SRC/esc/`.

[32] William R. Cook, Walter L. Hill, and Peter S. Canning. *Theoretical Aspects of Object-Oriented Programming*, chapter 14 - Inheritance Is Not Subtyping, pages 497–517. The MIT Press, 1994.

[33] Thierry Coquand and Guo-Qiang Zhang. Sequents, frames, and completeness. In *Proceedings of the 14th Annual Conference of the EACSL on Computer Science Logic*, pages 277–291, Springer-Verlag, 2000.

[34] David Crocker. Perfect Developer: A tool for object-oriented formal specification and refinement. In FME 2003, Tools Exhibition Notes, `http://www.eschertech.com/papers/fme_2003_tools_paper.pdf`, 2003.

[35] David Crocker. Safe object-oriented software: the verified design-by-contract paradigm. *Proceedings of the Twelfth Safety-Critical Systems Symposium*, pages 19–41, 2004.

[36] Willem Paul de Roever and Kai Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1998.

[37] David Detlefs, Greg Nelson, and James B. Saxe. Simplify: A theorem prover for program checking. Technical Report TR-15-81, August 05 2003.

[38] E. Dürr and J.v. Katwijk. VDM++, A Formal Specification Language for Object Oriented Designs. In *COMP EURO 92*, pages 214–219. IEEE, May 1992.

[39] Katherine A. Eastaughffe. Support for interactive theorem proving: Some design principles and their application, May 31 1998. `http://www.cl.cam.ac.uk/users/kae24/principles.ps.gz`.

[40] E.H. Dürr and N. Plat. VDM++ language reference manual. Technical Report ESPRIT-III project number 6500) document AFRO/CG/ED/LRM/V9, CAP Gemini Innovation, 1994.

[41] John S Fitzgerald. *Modularity in Model-Oriented Formal Specifications and its Interaction with Formal Reasoning.* Ph.D. thesis, University of Manchester, Computer Science Department, November 1991.

[42] David Stoutamire Gilad Bracha, Martin Odersky and Philip Wadler. Making the future safe for the past: Adding genericity to the java programming language. *OOPSLA 98, ACM*, 1998.

[43] Mike Gordon. HOL - A machine oriented formulation of higher order logic, January 12 2001. `http://www.cse.ogi.edu/~mjcg/Lectures/HOL.ps`.

[44] A. Grau. *Computer-Aided Validation of Formal Conceptual Models.* Ph.D. thesis, Technical University Braunschweig, Germany,, 2001.

[45] The VDM Tool Group. VDM++ Toolbox User Manual. Technical report, IFAD, October 2000. ftp://ftp.ifad.dk/pub/vdmtools/doc /usermanpp_letter.pdf.

[46] Joseph Y. Halpern and Moshe Y. Vardi. Model checking vs. theorem proving: A manifesto. In J. Allen, R. E. Fikes, and E. Sandewall, editors, *Proceedings 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning, KR'91*, pages 325–334. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[47] David Harel. *First-order dynamic logic.* PhD thesis, MIT, 1978.

[48] Christian Heinlein. APPLE: Advanced procedural programming language elements, April 20 2004. http://prog.vub.ac.be/~wdmeuter/PostJava04/papers/Heinlein.pdf.

[49] C. A. R. Hoare. An axiomatic basis for computer programming. In Manfred Broy and Ernst Denert, editors, *Software Pioneers — Contributions to Software Engineering*, pages 367–383, Berlin-Heidelberg-New York-Barcelona-Hong Kong-London-Milan-Paris-Tokyo, 2002. Springer-Verlag.

[50] C. A. R. Hoare. Communicating Sequential Processes (reprint). *Communications of the ACM*, 26(1):100–106, ACM, 1983.

[51] Chris Hostetter. Survey of object-oriented programming languages. http://www.rescomp.berkely.edu/~hossman/cs263/paper.html, May 1998.

[52] J. Hsiang and L. Bachmair. *Rewrite Method in Theorem Proving*. Academic Press, 1991.

[53] P. Hudak and J. H. Fasel. A gentle introduction to haskell. *SIGPLAN Notices*, 27(5), May 1992.

[54] Multi-Edit Software Inc. Multi-edit software inc. http://www.multiedit.com/.

[55] Paola Inverardi. On Relating Algebraic Specifications To VDM Specifications. In *Proc. of Nyborg Conf. on Combining Specifications*, May 1984.

[56] Daniel Jackson. *Alloy 3.0 Reference Manual*, 1.0 edition, May 2004. http://alloy.mit.edu/reference-manual.pdf.

[57] Michael Jackson. *Software Requirements & Specifications*. Addison-Wesley, 1995.

[58] R. Juellig, Y. Srinivas, and J. Liu. SPECWARE: An advanced environment for the formal development of complex software systems. *Lecture Notes in Computer Science*, 1101, Springer, 1996.

[59] Ingyu Kang. Crimson editor. `http://www.crimsoneditor.com/`.

[60] R. A. Kemmerer. Testing formal specifications to detect design errors. *IEEE Transactions on Software Engineering*, 11(1):32–42, 1985.

[61] James C. King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, July 1976.

[62] Joseph Roland Kiniry. *Kind Theory*. PhD thesis, Department of Computer Science, California Institute of Technology, May 10 2002.

[63] Sentot Kromodimoeljo, Bill Pase, Mark Saaltink, Dan Craigen, and Irwin Meisels. *The EVES System*, volume 693 of *Lecture Notes in Computer Science*, pages 349–373. Springer Verlag, 1993.

[64] L. Lamport. The Temporal Logic of Actions. Technical Report 79, Digital Equipment Corporation, Systems Research Centre, December 1991.

[65] Gary T. Leavens, Albert L. Baker, and Clyde Ruby. Preliminary design of JML: a behavioral interface specification language for Java. Technical Report 98-06q, Department of Computer Science, Iowa State University, 2001.

[66] Mícheál Mac an Airchinnigh. Mathematical Structures and their Morphisms in Meta IV. In Dines Bjørner, Cliff B. Jones, Mícheál Mac an Airchinnigh, and Eric J. Neuhold, editors, *VDM'87, VDM — A Formal Method at Work*, number 252 in Lecture Notes in Computer Science, pages 287 – 320, Berlin, 1987. Springer-Verlag.

[67] C. Marche, C. Paulin Mohring, and X. Urbain. The KRAKATOA tool for certification of JAVA/JAVACARD programs annotated in JML. *Journal of Logic and Algebraic Programming*, 58(1–2):89–106, 2004.

[68] Claude Marche and Christine Paulin-Mohring. Reasoning on java programs with aliasing and frame conditions. `http://krakatoa.lri.fr/papers/krakatoa-modele-draft.ps`, 2004.

[69] James L. Mcdonald and Yellamraju V. Srinivas. The architecture of SPECWARE, a formal software development system, May 10 1996. `ftp://ftp.kestrel.edu/pub/papers/specware/specware-arch.ps.Z`.

[70] Irwin Meisels, Ora Canada, and The Z Browser. Software manual for windows Z/EVES version 1.5 and the Z browser. http://citeseer.ist.psu.edu/15817.html; ftp://ftp.ora.on.ca/pub/doc/97-5505-04e.ps.Z, December 04 1996.

[71] B. Meyer. *Eiffel: An Introduction*. Interactive Software Eng., June 1988.

[72] Bertrand Meyer. Applying Design by Contract. *IEEE Computer*, 25(10):40–51, October 1992.

[73] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall, 1997.

[74] Tim Miller and Paul Strooper. Model-based specification animation using test-graphs. *Lecture Notes in Computer Science*, 2495, Springer, 2003.

[75] MIT Software Design Group. The Alloy Analyzer homepage. `http://alloy.mit.edu/`.

[76] Wojciech Mostowski. Formalisation and verification of Java Card security properties in Dynamic Logic. In Maura Cerioli, editor, *Proceedings, Fundamental Approaches to Software Engineering (FASE) Conference 2005, Edinburgh, Scotland*, volume 3442 of *LNCS*, pages 357–371. Springer, April 2005.

[77] Ataru Nakagawa and Kokichi Futatsugi. An overview of cafe project, October 23 1997. `http://www.ldl.jaist.ac.jp/cafeobj/papers/overview.ps.gz`.

[78] Eric Norman. LISP primer. Technical report, Academic Computing Center Madison, Wisconsin, 1975. `http://www.frobenius.com/primer.htm`.

[79] Martin Odersky and Philip Wadler. Pizza into Java: Translating theory into practice. Technical Report 26/96, University of Karlsruhe, July 1996.

[80] Security of Software Group at University of Nijmegen. SoS Research - ESC/Java2 Project. `http://www.cs.kun.nl/sos/research/escjava/main.html`.

[81] Jonathan S. Ostroff and Richard F. Paige. Developing BON as an industrial-strength formal method, October 01 1999.

[82] S. Owre, J. M. Rushby, , and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, 15-18 June 1992. Springer-Verlag. URL: `http://www.csl.sri.com/papers/cade92-pvs/`.

[83] Bill Pase, Dan Craigen, Irwin Meisels, Mark Saaltink, and Sentot Kromodimoeljo. A tutorial on EVES using s-Verdi, June 27 1995. `ftp://ftp.ora.on.ca/pub/doc/95-6018-60.ps.Z`.

[84] Benjemin C. Pierce. *Types and Programming Languages*. The MIT Press, 2002.

[85] Alain Pirotte. Automatic theorem proving based on resolution. *Annual Review in Automatic Programming*, 7(4):201–266, 1973.

[86] Amir Pnueli. System specification and refinement in temporal logic, November 08 1995. `ftp://ftp.wisdom.weizmann.ac.il/pub/amir/india92.ps.gz`.

[87] Thomas E. Potok, Mladen Vouk, and Andy Rindos. Productivity analysis of object-oriented software developed in a commercial environment. *Software Practice and Experience*, 29(10):833–847, August 1999.

[88] Zhenyu Qian. Combining object-oriented and functional language concepts. *Journal of Software*, 11(1):8–22, 2000.

[89] Gordon Rose and Graeme Smith. *The Object-Z Specification Language: Advances in Formal Methods Series*. Kluwer Academic Publishers, 2000.

[90] Amokrane Saibi, Benjamin Werner, Bruno Barras, Catherine Parent, Chetan Murthy, Christine Paulin-mohring, Cristina Cornes, Hugo Herbelin, Jean

christophe Filliatre, Judicael Courant, Projet Coq, and Samuel Boutin. The coq proof assistant - reference manual version 6.1, May 06 1997. ftp://ftp.inria.fr/INRIA/publication/publi-ps-gz/RT/RT-0203.ps.gz.

[91] Steve Schneider. *Concurrent and Real-time Systems: The CSP Approach*. Wiley, 1999.

[92] Steve Schneider and Helen Treharne. Verifying controlled components. In *Proc. IFM*, 2004.

[93] Abteilung Semantik, Clemens Fischer, and Fachbereich Informatik. CSP-OZ: A combination of object-Z and CSP, November 12 1997.

[94] Siemens. Siemens transport system. http://www.siemens-ts.com/pagesUS/produits/Meteor.htm.

[95] Karli Watson Jay Glynn Morgan Skinner Bill Evjen Simon Robinson, Christian Nagel. *Professional C#*. Wrox, 2002.

[96] Colin Snook and Michael Butler. Verifying dynamic properties of UML models by translation to the B language and toolkit. *Proc. UML 2000: Advancing the standard*, LNCS Vol. 1939, A. Evans, S. Kent, B. Selic (editors), Springer-Verlag, 2000.

[97] Colin Snook and Rachel Harrison. Practitioners' views on the use of formal methods: an industrial survey by structured interview. March 01 2001.

[98] Jose H. Solorzano and Suad Alagić. Parametric polymorphism for Java: A reflective solution. *ACM SIGPLAN Notices*, 33(10):216–225, October 1998.

[99] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall, 2nd edition, 1992.

[100] Rajat R. Sud and James D. Arthur. Requirements management tools: A quantitative assessment. Technical Report 03-10, February 01 2003.

[101] Simon Thompson. *Haskell: The Craft of Functional Programming*. Addison-Wesley, Harlow, England, 1996.

[102] Mads Torgersen. Inheritance is specialization. *Proc. Object Oriented Technology. ECOOP 2002 Workshop and Posters, LNCS vol. 2548*, Juan Hernandez, Ana M.D. Moreira (editors), Springer-Verlag, 2002.

[103] Mark Utting. Reasoning about aliasing, September 24 1997. `http://www.cs.waikato.ac.nz/~marku/papers/aliasing.ps.gz`.

[104] Myra Vaninwegen. HOL-ml, January 16 0. `ftp://ftp.cis.upenn.edu/pub/papers/myra/holML.ps`.

[105] Markus Wenzel and Tu Munchen. The isabelle/isar reference manual, May 10 2000. `http://isabelle.in.tum.de/PSV2000/doc/isar-ref.pdf`.

[106] Thomas Wilson and Savi Maharaj. Omnibus: A clean language for supporting dbc, esc and vdbc. `http://www.cs.stir.ac.uk/~twi/docs/fm05.pdf`, 2005.

[107] Jeannette M. Wing. A study of 12 specifications of the library problem. *IEEE Softw.*, 5(4):66–76, 1988.

[108] M. Zhang, P. Scheyen, Q. Zhuang, and S. Yu. Introducing KINDs to C++, December 13 1995. `http://www.csd.uwo.ca/~pete/kc.ps`.