ISICL Report

Michele Cozart
Matthew Keitz
Michael Marcus

May 14, 2003

Contents

Chapter 1

An Introduction to ISICL

The Intuitive Strategic Intelligence Control Language (ISICL, pronounced "icicle") provides a straightforward method for defining and controlling simulated robotic tanks in Robocode, a real-time strategy game created by IBM. ISICL allows avid Robocoders to design their robots around the model of a finite state machine, which is more in keeping with standard paradigms in artificial intelligence programming. The compiler does the grunt work of translating the programmer's ideas into a fully functional Robocode robot, leaving the programmer free to manipulate concepts rather than code.

1.1 Background

Robocode is an easy-to-use robotic battle simulator created by Matthew Nelson, a software engineer at IBM. In Robocode, users are encouraged to create their own customized robots within the framework of a specialized Java API. Indeed, the primary purpose of Robocode is to provide a fun, interesting and challenging way for novice programmers to learn the Java language. Since the release of version 1.0 in April 2002, Robocode has expanded to provide a common arena for programmers at all experience levels to hone their skills.

The robots are simulated in the Robocode GUI as autonomous robotic tanks. Each tank is equipped with a gun and radar, which can rotate independently of the robot and of each other. The robot itself can move in any direction, either going forward or in reverse. Robots begin every battle with 100 life energy points, which are lost when it incurs damage (i.e., when it is

hit by another robot's bullet or when it collides with one of the walls). A robot dies when it runs out of life energy. The objective is to be the last robot alive at the end of the round.

For more information, please visit the Robocode website at http://robocode.alphaworks.ibm.com/.

1.2 The Motivation for ISICL

ISICL is an intuitive, portable, and extensible language for strategic intelligence control in Robocode. It is designed to simplify the process of creating a Robocode robot for the express purpose of competition. We present ISICL as a viable alternative to the more cumbersome method of Java programming within the Robocode domain.

1.2.1 Intuitive

The concept of a finite state machine is intrinsic to Robocode but only hinted at by the Java implementations of its robots. The Java language, though not without its advantages, is extremely broad in scope. As a language it is ill-suited to the task of programming artificial intelligence because so many implementation details, such as event handling, are left to the programmer.

ISICL offers a higher level of abstraction than Java, leaving the programmer free to focus on the *what* and not the *how* of constructing a robot. The state-machine model is readily applicable to a Robocode battle: every robot begins in an initial state, transitioning to other states — a state of high alert, for example — when certain battle conditions are met. By specifying states and state transitions, while allowing the ISICL compiler to implement the handling of Java events, the programmer is able to more readily translate ideas into code. Common English keywords make the code itself easy to learn as well.

1.2.2 Portable

Because ISICL compiles into Java source code, it inherits the portability and architectural independence of the Java virtual machine. Robocode is currently available on Windows, Macintosh and Unix platforms; a program

written in ISICL can be compiled on any of those platforms and run seamlessly on any of the others.

1.2.3 Extensible

Extensibility is a key aspect of the ISICL design. Version 1.0 implements a small, robust language that can be easily expanded to include increased functionality. A standard header file includes definitions of the most common Robocode commands and mathematical functions, each with a corresponding Java implementation. ISICL relies heavily on Java inheritance, so that this runtime library can be readily expanded with straightforward implementations of a standardized Java interface. By using inheritance in developing our language, we were able to reduce redundant code and keep the syntax simple while allowing maximum flexibility in what the ISICL programmer can describe.

1.2.4 Strategic Intelligence Control

The competitive aspect of Robocode is the driving force behind its popularity, and strategy lies at the forefront of that competition. A successful Robocode battle strategy involves knowing how to respond to various events and battle conditions. A successful Robocode programmer should therefore be focused on strategic concepts more so than having to execute those concepts. ISICL's state-machine design is ideal for this programming methodology.

Chapter 2

Tutorial

ISICL allows the programmer to specify a Robocode robot in terms of a finite state machine. A state is defined by an action followed by an arbitrary number of outgoing transitions. Each transition consists of a condition that, when true, causes the robot to transition to the corresponding state. The conditions are evaluated in the order in which they are specified.

2.1 A Simple Robot

To begin, we will create a simple robot that just moves back and forth until the game ends. Figure 2.1 shows the state diagram for this robot.

This robot is defined by only two states — a "moving forward" state and a "moving backward" state. The corresponding ISICL code is very straightforward:

```
state START:
ahead 5
[
        true -> MOVEBACK
]

state MOVEBACK:
back 5
[
        true -> START
]
```

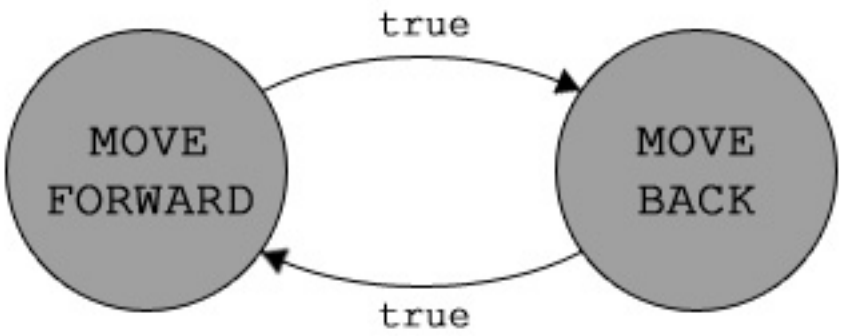

Figure 2.1: A robot that moves back and forth.

Let's examine this code line by line.

The first line, `state START`, declares the start state for the robot. In ISICL, every robot must have a start state, and it must be labeled as such (this is why we did not call the state `MOVEUP`). We use capital letters to label the states only as a convention to offset state names from other identifiers.

The next line, `ahead 5`, tells the robot to move 5 units in the direction it is facing (called the "heading"). `ahead` is a predefined action in ISICL; take a look at the next section for a complete list.

The next three lines comprise the transition set for the `START` state, indicated by the left and right square brackets. Here we have only one transition: `true -> MOVEBACK`. This means, "When the condition `true` is true, transition to state `END`." Since `true` is always true, this robot will always transition to the `MOVEBACK` state after it performs the action for the `START` state.

The `MOVEBACK` state is constructed in exactly the same fashion, except we want the robot to move backward 5 units and transition back to the `START` state. That's all there is to it!

2.2 Compiling and Running an ISICL Robot

Now that we have created a robot, we need to save and compile it. All ISICL files must end with a ".icl" extension; we'll call this one firstExample.icl.

Save the file to the parent directory of the ISICL packages, and type:

```
$ java isicl.Main firstExample.icl
```

This produces a file called FirstExample.java, which you need to run through the Java compiler.

```
$ javac FirstExample.java
```

This creates FirstExample.class, a Java bytecode file that Robocode can run. For details on how to load your robot into Robocode, please see the Robocode documentation.

2.3 A More Complicated Example

Let's look at a robot that does something a bit more interesting. The robot represented by the state diagram in Figure 2.2 moves around and shoots at the other robots that it detects any on its radar.

Even though the state diagram is a bit more involved, it is no more difficult to implement this robot in ISICL. The code is given below.

```
state START:
        pass
        [
                coin_toss -> SPIN_LEFT
                true -> SPIN_RIGHT
        ]

state SPIN_LEFT:
        left 1
        [
                scanned_enemy -> SHOOT
                panic or stop_turning -> MOVE
                true -> SPIN_LEFT
        ]
```

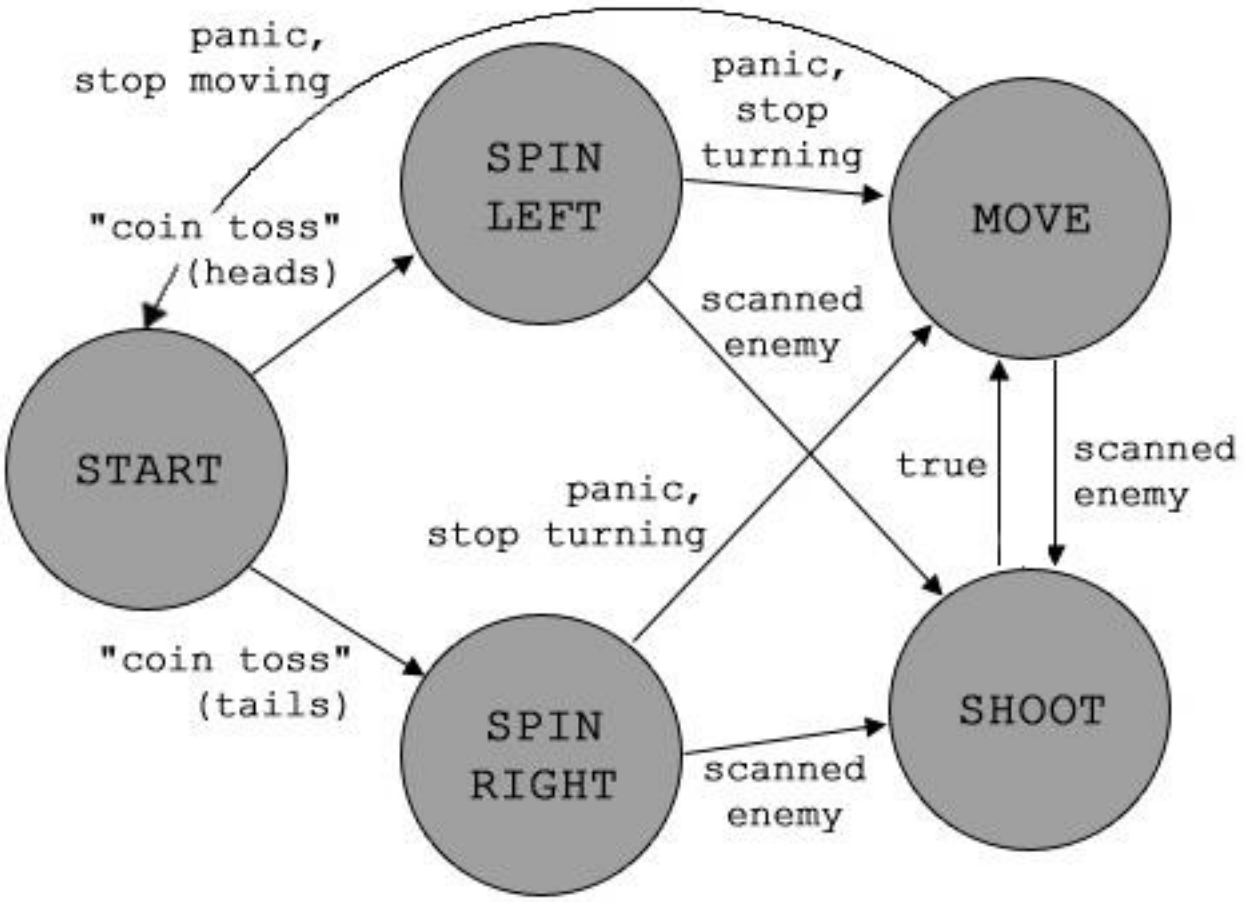

Figure 2.2: A robot with more interesting behavior. Reflexive transitions have been omitted for clarity.

```
state SPIN_RIGHT:
        right 1
        [
                scanned_enemy -> SHOOT
                panic or stop_turning -> MOVE
                true -> SPIN_RIGHT
        ]

state SHOOT:
        fire 1
        [
                true -> MOVE
        ]

state MOVE:
        ahead 1
        [
                scanned_enemy -> SHOOT
                panic or stop_moving -> START
                true -> MOVE
        ]

condition coin_toss: random 0.5

condition stop_moving: random 0.05

condition stop_turning: random 0.01

condition panic: robot_died or hit_self
```

The robot's action in the `START` state is `pass`, which means that no action is taken. From the `START` state the robot will transition to the `SPIN_LEFT` state when `coin_toss` is true and the `SPIN_RIGHT` state otherwise. In `SPIN_LEFT` the robot turns one unit to the left, then transitions to the `SHOOT` state if it spots an enemy, the `MOVE` state if `panic` or `stop_turning` is true, or back to itself otherwise. (We will come to the meanings of `coin_toss`, `panic` and `stop_turning` shortly.) The `SPIN_LEFT` state is similar, except the robot turns right instead of left. The `SHOOT` state consists of a command to fire

the gun with a power level of 1, and an unconditional transition to the `MOVE` state. The `MOVE` state moves the robot forward 1 unit and transitions to `SHOOT` if it spots an enemy, `START` if `panic` or `stop_moving` is true, and `MOVE` otherwise.

There are two important things to notice about this robot. First, we have declared multiple outgoing transitions for some of the states, whereas the previous example had only one transition per state. You can have as many transitions as you like for a given state, as long as they are enumerated in between the square brackets. The second thing to notice is that we have defined the custom conditions `coin_toss`, `stop_moving`, `stop_turning` and `panic` at the bottom of the file. We use the predefined function `random`, which returns `true` with a probability equal to its argument, to create a robot that exhibits probabilistic behavior. `coin_toss` simulates the flip of a fair coin; it returns `true` half the time and `false` the other half. Similarly, `stop_moving` will return `true` with probability 0.05, and `stop_turning` will return `true` with probability 0.01. The final custom condition, `panic`, returns true if the predefined conditions `robot_died` or `hit_self` are true. (In addition to conditions, you can also define your own custom actions and functions in ISICL. See the 3.4.2 and 3.4.3 declarations for details.)

Chapter 3

Language Reference

3.1 Notation

In this document, *italic* text represents a non-terminal symbol; `fixed-width` text represents one or more terminal symbols. In grammar specifications, a non-terminal followed by colon (":") denotes the left side of a production; each of the lines that follow, up to the next production, represent different strings that may be derived from that symbol. All symbols appearing in such strings are terminal, with the exception of italicized non-terminals. A non-terminal with the subscript "*opt*" indicates an optional clause. If the left hand side of a production is followed by the words "one of", the line or lines that follow contain a space-separated list of alternative productions.

3.2 Program Structure

ISICL is an alternative robot control language for the Robocode robot combat simulator. (See http://robocode.alphaworks.ibm.com for details on the Robocode API and environment.) ISICL defines a robot's actions in terms of states. All robots begin in the `START` state. After the robot has finished executing the action associated with the current state, it transitions to a new state, chosen according to the transition expressions listed in the current state. When this transition occurs, a command is issued to the Robocode API; this is the only time the ISICL programmer can make changes to the environment.

3.3 Language Elements

3.3.1 Whitespace

Spaces, tabs, line feeds, form feeds, and carriage returns are significant in ISICL only when necessary to separate identifiers in an argument or transition list, or to close a comment (see 3.3.2.) In all other cases, whitespace characters are ignored.

3.3.2 Comments

Any text appearing between two forward slashes ("`//`") and the end of a line is considered a comment, and ignored by the compiler.

3.3.3 Literals

number:
    $sign_{opt}$ *digits* $frac_{opt}$ exp_{opt}
    $sign_{opt}$ *frac* exp_{opt}

frac:
    `.` *digits*

sign: one of
    `+ -`

exp:
    `e` *digits*
    `E` *digits*

digits:
    digit
    digit *digits*

digit: one of
    `0 1 2 3 4 5 6 7 8 9`

All literals in ISICL are floating point numbers. In a logical context, zero is considered `false`; any other value is `true`.

3.3.4 Identifiers

identifier:
    letter *letters-and-digits*$_{opt}$

letters-and-digits:
    letter *letters-and-digits*$_{opt}$
    digit *letters-and-digits*$_{opt}$
    _ *letters-and-digits*$_{opt}$

letter: one of
    `a b c d e f g h i j k l m n o p q r s t u v w x y z`
    `A B C D E F G H I J K L M N O P Q R S T U V W X Y Z`
    `_ $`

The name of a state, action, or function must begin with a letter, and contain only letters, numbers, dollar signs ("$"), and/or underscores ("_"). Identifiers are not case sensitive. Function and action identifiers share the same namespace; it is an error to give an action or function the same name as another action or function in the same program, or in the library of predefined actions and functions.

3.3.5 Expressions

expression:
    expression `and` *comp-expression*
    expression `or` *comp-expression*
    expression `xor` *comp-expression*
    `not` *comp-expression*
    comp-expression

comp-expression:
    num-expression `=` *num-expression*
    num-expression `!=` *num-expression*
    num-expression `>=` *num-expression*
    num-expression `<=` *num-expression*
    num-expression `>` *num-expression*
    num-expression `<` *num-expression*
    num-expression

num-expression:
    num-expression `+` *term-expression*
    num-expression `-` *term-expression*
    term-expression

term-expression:
    term-expression `*` *value-expression*
    term-expression `/` *value-expression*
    value-expression

value-expression:
    number
    identifier *arguments*$_{opt}$
    `(` *expression* `)`

arguments:
    `(` *expression-list*$_{opt}$ `)`

expression-list:
    expression *more-expressions*$_{opt}$

more-expressions:
    `,` *expression* *more-expressions*$_{opt}$

Expressions in ISICL follow standard mathematical notation for arithmetic (addition, subtraction, multiplication, and division,) comparisons (equal, not equal, less than, greater than, less than or equal, and greater than or equal,) and logical operations (and, or, xor, and not.) Expressions operate on numeric literals and/or function calls. Arguments are passed to functions as a list of expressions, surrounded by parentheses; if the function takes no arguments, the empty parentheses may be omitted.

3.4 Declarations

An ISICL program is a list of declarations. The order in which declarations are processed is not defined, and a particular declaration statement may refer to any state, function, or action defined in the program, including itself. (Note, however, that ISICL expressions may or may not be "short-circuit" evaluated, so a condition that refers to itself is unlikely to be useful.)

3.4.1 `state`

state:
`state` *identifier*: *action* [*transition-list*$_{opt}$]

action:
identifier *expression-list*$_{opt}$

transition-list:
transition *transition-list*$_{opt}$

transition:
expression `->` *identifier*

Defines a state. When a state transition occurs, each transition associated with the current state is evaluated, in order, until a transition expression returns `true`; the state indicated on the right side of that transition then becomes the current state, and the action associated with the new state is executed. The new state may be the same as the current state. If, due to a runtime error, an expression fails to generate a meaningful result, it is treated as `false`. If no transition expression returns `true`, the robot returns to the `START` state. (This is a "fail safe".)

3.4.2 `action`

action:
`action` *identifier*: *identifier*

Defines a robot action, to be used in the *action* clause of a state declaration. The second *identifier* must be the name of a java class that will be included during the compilation process (in an implementation-defined fashion.) The class must implement the `isicl.interfaces.ActionImplementation` interface (see 3.7.1.)

3.4.3 `function`

function:
`function` *identifier*: *identifier*

Defines a mathematical or informational function, to be used in expressions. The second *identifier* must be the name of a java class that will be included during the compilation process (in an implementation-defined fashion.) The

class must implement the `isicl.interfaces.FunctionImplementation` interface (see 3.7.2.) Some implementations may optimize generated code by displacing or eliminating function calls, so functions must be referentially transparent. If a function makes any changes to the state of the robot or the environment, the resulting behavior is undefined.

3.4.4 `condition`

condition:
    `condition` *identifier*: *expression*

Stores a valid expression, to be used elsewhere in the program. When an expression that includes the name of a condition as a subexpression is evaluated, the stored expression is evaluated in the current context, and its result becomes the value of the subexpression. In effect, this declaration creates a zero-argument function that computes the indicated expression.

3.4.5 `const`

const:
    `const` *identifier*: *literal*

Stores a constant, which can then be used in place of a literal. In effect, this declaration creates a zero-argument function that always returns the specified value.

3.5 Predefined Actions

The ISICL standard library includes several predefined actions. They are declared in the "standard.icl" header file, and implemented by the `isicl.runtime.actions` package.

`ahead` *distance*

`back` *distance*

Moves your robot forward or backward, respectively, until *distance* is reached, the `stop` action is issued, or an obstacle is encountered.

`right` *degrees*

`left` *degrees*

Turns your robot *degrees* to the right or left of its current heading.

`stop`
Cancels any movement in progress.

`resume`
Resumes the last movement that was left incomplete due to a `stop` action.

`fire` *power*
Fires your robot's weapon, in the direction that the robot is currently facing, at the indicated power level. *power* should be between 0 and 3.

`pass`
No action is taken, and a new state transition is triggered immediately.

3.6 Predefined Functions

3.6.1 Flags

All of the following functions return information about events that have occurred since the last time a transition was triggered.

`hit_enemy`
Returns `true` if your robot's weapon hit an enemy.

`hit_bullet`
Returns `true` if a bullet from your robot's weapon hit another bullet.

`missed`
Returns `true` if a bullet from your robot's weapon hit a wall.

`hit_self`
Returns `true` if your robot was struck by a bullet.

`bump_enemy`

Returns `true` if your robot tried to move into a space occupied by another robot, or vice versa.

`bump_wall`
Returns `true` if your robot tried to move into a wall.

`scanned_enemy`
Returns `true` if a robot has been sighted.

`robot_died`
Returns `true` if a robot died.

3.6.2 Information Functions

These functions return information about the robot and its environment.

`xpos`
`ypos`
Your robot's current position on the map.

`heading`
Your robot's current heading, in degrees.

`speed`
Your robot's current speed.

`energy`
Your robot's current energy level.

`map_height`
`map_width`
The dimensions of the battlefield.

`enemies`
The number of enemy robots left on the battlefield.

`random` *chance*
chance is a value between 0 and 1. Returns `true` *chance* out of 1 times.

3.6.3 Sensor Functions

These functions return information about the last enemy that was scanned by the robot's radar.

`enemy_direction`
The direction from your robot to the enemy scanned, in degrees.

`enemy_distance`
Distance to the enemy.

`enemy_energy`
Energy level (health) of the enemy.

`enemy_heading`
The enemy's current heading.

3.6.4 Utility Functions

These functions are intended for use in mathematical expressions, and have their conventional meaning.

`true`
`false`
`sin` *degrees*
`cos` *degrees*
`tan` *degrees*
`floor` *number*
`pow` *base exp*

3.7 Interfaces

These interfaces can be found in the `isicl.interfaces` package.

3.7.1 ActionImplementation

A Java class that is used to define an action must implement this interface. When a state transition causes that action to be executed, the `callAction` method will be invoked, with a reference to your robot as the first argument, and an array of parameters as the second argument. `getNumArgs` should return the number of arguments that the action expects.

```
public interface ActionImplementation {
        public abstract void
                callAction(Robot bot, double args[]);

        public abstract int
                getNumArgs();
}
```

3.7.2 FunctionImplementation

A Java class that is used to define a function must implement this interface. When the function is called in a transition expression, or in an argument expression in an action call, the `callFunction` method will be invoked, with a reference to the robot as the first argument, and an array of parameters as the second argument. The function should return a value representing the result of the computation. `getNumArgs` should return the number of arguments that the function expects.

```
public interface FunctionImplementation {
        public abstract double
                callFunction(final Robot bot, double args[]);

        public abstract int
                getNumArgs();
}
```

Chapter 4

Project Plan

4.1 General Procedures

The process we used for planning, specification, development, and testing was fairly relaxed. We were a small group and the project was not terribly long or complex. We therefore felt that implementing heavy processes would take more time that the benefit wed derive from them would be worth.

During the entire project, we met twice weekly via AOL Instant Messenger (AIM). At the beginning of the project, we used these meeting times to spec out the project and assign preliminary tasks. Since communication only via IM can be very time consuming, we decided on the project idea and goals early on. Matt and Mark took charge of language design, conducting independent meetings in person, via email and over IM to flesh out their ideas. After they had a reasonable idea of what the design would be, we took time during a bi-weekly meeting to discuss and approve their design.

Due dates on the whitepaper, language reference manual and final paper also helped to structure our project. We used these dates to help guide us as to where we should be on the project at key junctures during the semester. Meeting with Dr. Edwards regularly also helped keep the schedule on track.

During the last week of the project, we had daily IM meetings and kept our IM clients open in case of difficulties, which helped us be as productive as possible. On several occasions, one or the other of us would need something and be able to retrieve it instantly, which enabled us to return to work more readily.

The three remaining members of the group pitched in wherever help was

needed, especially during the critical final weeks. The testing process included four types of tests: unit testing, regression testing, ad hoc and integration testing. The unit testing was done by the individual engineers. It was their responsibility to test their code before it was checked in and to make sure that their code worked with the code already in the repository. Regression testing was performed by the individual engineers. Ad hoc testing was done by all of the engineers. The integration testing was the responsibility of the test engineer and was designed to locate bugs that prevented end-to-end functionality from working.

Defects were tracked via e-mail, since there were not a large number and we did not have time to test as thoroughly as we had planned. The person who was most familiar with the area of defective code would work on repairing that defect. The planned process for defect tracking was going to be handled on a web page, but we never reached a point in the project where this would have been useful.

4.2 Style Guide

Our team used normal Java styling for our code. We also used Javadocs to comment our classes and methods. Here is the style guide we used for our Javadocs.

4.2.1 Documenting a Class

```
/**
 * <p>Description: <p>
 *
 * @author
 * @version
 * @see
 * <p>Project:  ISICL  COMS4115</p>
 */
```

4.2.2 Documenting a Field

```
/**
 * Description:
 */
```

4.2.3 Documenting a Method

```
/**
 * Description:
 *
 * @param
 * @return
 * @exception
 * @see
 * Creation date:
 */
```

4.3 Project Timeline

Date	Topic of Discussion
February 9, 2003	Initial meeting intros, discuss project ideas
February 22, 2003	Language design meeting
February 25, 2003	White paper meeting with Professor Edwards
February 26, 2003	Language design meeting
March 2, 2003	Discuss language design
March 5, 2003	Discuss language design
March 9, 2003	Finalize language design and tasks
March 23, 2003	LRM Discussion meeting
March 25, 2003	LRM Review rough draft
March 26, 2003	LRM Discussion meeting with Professor Edwards
March 30, 2003	Discuss antlr tasks
April 3, 2003	Discuss lexer/parser progress
April 6, 2003	Discuss lexer/parser progress
April 13, 2003	Discuss lexer/parser disagreements
April 20, 2003	Discuss remaining tasks, made task assignments
April 30, 2003	Meet with Dr. Edwards to discuss our project status
May 4, 2003	Status updates, task confirmations, defects
May 9, 2003	Status updates
May 10, 2003	Status updates
May 11, 2003	Status updates
May 12, 2003	Informal IM status updates
May 13, 2003	Status updates, task coverage and final assignments

4.4 Roles

Michele Cozart	white paper, team website, organization and scheduling, testing, compiler entry point (Main.java)
Matthew Keitz	language design[1] , reference manual, lexer/parser semantic checking, symbol table, standard library, documentation layout
Michael Marcus	code generation, runtime design[1], runtime implementation, tutorial, revised white paper, version control

4.5 Tools

Java 1.4.02	implementation
ANTLR 2.7.2	parser/lexer generation
Robocode 1.0.6	target runtime library
TeX 3.141592 (MiKTeX 2.2)	documentation
RCS 5.7	version control

[1]Mark Berman contributed to the early stages of these tasks.

Chapter 5

Architectural Design

5.1 Block Diagram

A block diagram of our compiler is shown in Figure 5.1.

5.2 Interfaces

The interfaces between the compiler modules were very well-defined. Multiple layers of abstraction allowed us to develop each component independently.

Rather than including code generation modules in the tree walker itself, we devised Java objects pertaining to the various entitiies in our language that the tree walker would store in a package-accessible symbol table.

- **State**: An instance of State is created for every state declaration in the ISICL program. A State consists of Strings representing the state name, action (plus a Stack of arguments), and condition-transition pairs (parallel Stacks).
- **Action**: An instance of Action is created for every action encountered in the ISICL program, both pre- and user-defined. An Action consists of its name, a Stack of its arguments and its Java class implementation.
- **Function**: Similar to Action, but for ISICL functions.
- **UserFunction**: An instance of UserFunction is created for every condition and constant in the ISICL program. A UserFunction consists only of a Java expression that is evaluated by the generated code.

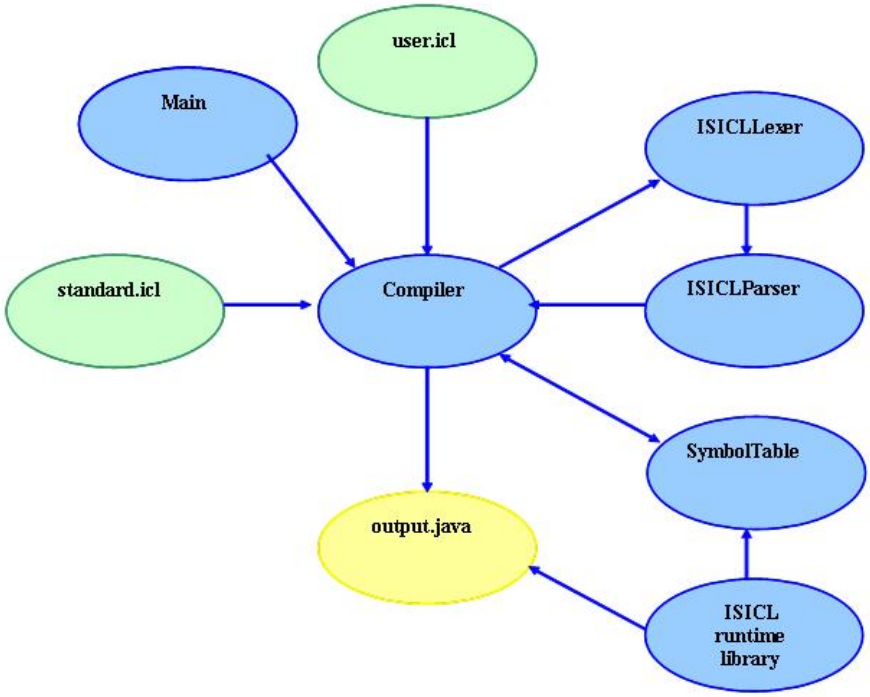

Figure 5.1: The architectural design of the ISICL compiler.

Matt was responsible for the initial implementation of these interfaces and the symbol table. Bug fixes and improvements were later added by the other members of the group.

Chapter 6

Test Plan

To test the translator, we created a test program that contained all of our predefined actions and functions to make sure that they were all recognized. We also created a test to verify all of our syntax including known syntax errors to make sure we caught them and that the error messages were appropriate.

The test cases we chose exercised expected syntax and semantic structure and unexpected syntax and semantic structure. We also tested end-to-end functionality to make sure that we were able to get a valid .java file our of our compiler.

We didn't use automation in our testing. Since the project was small and we were on a limited time budget, we chose to run the tests ourselves.

6.1 dumbRobot ISICL Code

```
state START:
        ahead
        [
                true -> End
        ]

state END:
        stop
        [
                true -> End
        ]
```

6.2 DumbRobot Java code

```
import isicl.runtime.*;
import java.util.*;

public class DumbRobot extends ISICLRobot {

        isicl.runtime.functions.Sin iclFunction_sin = new isicl.runtime.functions.Sin();
        public double _sin(double arg01) {
```

```
        double args[] = { arg01 };
        return iclFunction_sin.callFunction(this, args);
}

isicl.runtime.functions.True iclFunction_true = new isicl.runtime.functions.True();
public double _true() {
        double args[] = {  };
        return iclFunction_true.callFunction(this, args);
}

isicl.runtime.actions.Pass iclAction_pass = new isicl.runtime.actions.Pass();
public void _pass() {
        double args[] = {  };
        iclAction_pass.callAction(this, args);
}

isicl.runtime.functions.Cos iclFunction_cos = new isicl.runtime.functions.Cos();
public double _cos(double arg01) {
        double args[] = { arg01 };
        return iclFunction_cos.callFunction(this, args);
}

isicl.runtime.actions.Ahead iclAction_ahead = new isicl.runtime.actions.Ahead();
public void _ahead(double arg01) {
        double args[] = { arg01 };
        iclAction_ahead.callAction(this, args);
}

isicl.runtime.functions.False iclFunction_false = new isicl.runtime.functions.False();
public double _false() {
        double args[] = {  };
        return iclFunction_false.callFunction(this, args);
}

isicl.runtime.actions.Resume iclAction_resume = new isicl.runtime.actions.Resume();
public void _resume() {
        double args[] = {  };
        iclAction_resume.callAction(this, args);
}

isicl.runtime.functions.MapHeight iclFunction_map_height = new isicl.runtime.functions.MapHeight();
public double _map_height() {
        double args[] = {  };
        return iclFunction_map_height.callFunction(this, args);
}

isicl.runtime.functions.Xpos iclFunction_xpos = new isicl.runtime.functions.Xpos();
public double _xpos() {
        double args[] = {  };
        return iclFunction_xpos.callFunction(this, args);
}

isicl.runtime.functions.Pow iclFunction_pow = new isicl.runtime.functions.Pow();
public double _pow(double arg01, double arg02) {
        double args[] = { arg01, arg02 };
        return iclFunction_pow.callFunction(this, args);
}

isicl.runtime.functions.Heading iclFunction_heading = new isicl.runtime.functions.Heading();
public double _heading() {
        double args[] = {  };
        return iclFunction_heading.callFunction(this, args);
}

isicl.runtime.actions.Left iclAction_left = new isicl.runtime.actions.Left();
public void _left(double arg01) {
        double args[] = { arg01 };
        iclAction_left.callAction(this, args);
}

isicl.runtime.functions.Ypos iclFunction_ypos = new isicl.runtime.functions.Ypos();
public double _ypos() {
        double args[] = {  };
        return iclFunction_ypos.callFunction(this, args);
}

isicl.runtime.functions.MapWidth iclFunction_map_width = new isicl.runtime.functions.MapWidth();
```

```
public double _map_width() {
        double args[] = {  };
        return iclFunction_map_width.callFunction(this, args);
}

isicl.runtime.functions.Enemies iclFunction_enemies = new isicl.runtime.functions.Enemies();
public double _enemies() {
        double args[] = {  };
        return iclFunction_enemies.callFunction(this, args);
}

isicl.runtime.actions.Back iclAction_back = new isicl.runtime.actions.Back();
public void _back(double arg01) {
        double args[] = { arg01 };
        iclAction_back.callAction(this, args);
}

isicl.runtime.actions.Stop iclAction_stop = new isicl.runtime.actions.Stop();
public void _stop() {
        double args[] = {  };
        iclAction_stop.callAction(this, args);
}

isicl.runtime.functions.Speed iclFunction_speed = new isicl.runtime.functions.Speed();
public double _speed() {
        double args[] = {  };
        return iclFunction_speed.callFunction(this, args);
}

isicl.runtime.functions.Floor iclFunction_floor = new isicl.runtime.functions.Floor();
public double _floor(double arg01) {
        double args[] = { arg01 };
        return iclFunction_floor.callFunction(this, args);
}

isicl.runtime.functions.Energy iclFunction_energy = new isicl.runtime.functions.Energy();
public double _energy() {
        double args[] = {  };
        return iclFunction_energy.callFunction(this, args);
}

isicl.runtime.functions.Tan iclFunction_tan = new isicl.runtime.functions.Tan();
public double _tan(double arg01) {
        double args[] = { arg01 };
        return iclFunction_tan.callFunction(this, args);
}

isicl.runtime.actions.Right iclAction_right = new isicl.runtime.actions.Right();
public void _right(double arg01) {
        double args[] = { arg01 };
        iclAction_right.callAction(this, args);
}

isicl.runtime.actions.Fire iclAction_fire = new isicl.runtime.actions.Fire();
public void _fire(double arg01) {
        double args[] = { arg01 };
        iclAction_fire.callAction(this, args);
}


public void run() {
        s = new StateEngine("START");
        tv = new Vector();
        tv.add( new ConditionPair("END") { public boolean condition() {
                if( (_true()) != 0.0 ) return true;
                else return false;
        } });
        s.addState( new AbstractState("END", tv) { public void actions() { _stop(); } });


        tv = new Vector();
        tv.add( new ConditionPair("END") { public boolean condition() {
                if( (_true()) != 0.0 ) return true;
                else return false;
        } });
        s.addState( new AbstractState("START", tv) { public void actions() { _ahead((5)); } });
```

```
                while(true) {s.tick();}
        }
}
```

6.3 testBot ISICL Code

```
// Matthew Keitz

state START:
        pass
        [
                coin_toss -> SPIN_LEFT
                true -> SPIN_RIGHT
        ]

state SPIN_LEFT:
        left 1
        [
                scanned_enemy -> SHOOT
                panic or stop_turning -> MOVE
                true -> SPIN_LEFT
        ]

state SPIN_RIGHT:
        right 1
        [
                scanned_enemy -> SHOOT
                panic or stop_turning -> MOVE
                true -> SPIN_RIGHT
        ]

state SHOOT:
        fire 1
        [
                true -> MOVE
        ]

state MOVE:
        ahead 1
        [
                scanned_enemy -> SHOOT
                panic or stop_moving -> START
                true -> MOVE
        ]

condition coin_toss: random 0.5

condition stop_moving: random 0.05

condition stop_turning: random 0.01
```

6.4 TestBot Java Code

```
import isicl.runtime.*;
import java.util.*;

public class TestBot extends ISICLRobot {

        isicl.runtime.functions.BumpEnemy iclFunction_bump_enemy = new isicl.runtime.functions.BumpEnemy();
        public double _bump_enemy() {
                double args[] = {  };
                return iclFunction_bump_enemy.callFunction(this, args);
        }

        isicl.runtime.actions.Pass iclAction_pass = new isicl.runtime.actions.Pass();
        public void _pass() {
                double args[] = {  };
                iclAction_pass.callAction(this, args);
        }
```

```
isicl.runtime.functions.Cos iclFunction_cos = new isicl.runtime.functions.Cos();
public double _cos(double arg01) {
        double args[] = { arg01 };
        return iclFunction_cos.callFunction(this, args);
}

public double _panic() { return (((0 != (_robot_died())) || ((_hit_self()) != 0))? 1 : 0); }

isicl.runtime.actions.Left iclAction_left = new isicl.runtime.actions.Left();
public void _left(double arg01) {
        double args[] = { arg01 };
        iclAction_left.callAction(this, args);
}

isicl.runtime.functions.Floor iclFunction_floor = new isicl.runtime.functions.Floor();
public double _floor(double arg01) {
        double args[] = { arg01 };
        return iclFunction_floor.callFunction(this, args);
}

public double _true() { return (1); }

isicl.runtime.functions.ScannedEnemy iclFunction_scanned_enemy = new isicl.runtime.functions.ScannedEnemy();
public double _scanned_enemy() {
        double args[] = {  };
        return iclFunction_scanned_enemy.callFunction(this, args);
}

isicl.runtime.actions.Resume iclAction_resume = new isicl.runtime.actions.Resume();
public void _resume() {
        double args[] = {  };
        iclAction_resume.callAction(this, args);
}

isicl.runtime.functions.BumpWall iclFunction_bump_wall = new isicl.runtime.functions.BumpWall();
public double _bump_wall() {
        double args[] = {  };
        return iclFunction_bump_wall.callFunction(this, args);
}

isicl.runtime.functions.Sin iclFunction_sin = new isicl.runtime.functions.Sin();
public double _sin(double arg01) {
        double args[] = { arg01 };
        return iclFunction_sin.callFunction(this, args);
}

isicl.runtime.functions.Ypos iclFunction_ypos = new isicl.runtime.functions.Ypos();
public double _ypos() {
        double args[] = {  };
        return iclFunction_ypos.callFunction(this, args);
}

public double _stop_turning() { return (_random((0.01))); }

isicl.runtime.functions.EnemyDirection iclFunction_enemy_direction = new isicl.runtime.functions.EnemyDirection();
public double _enemy_direction() {
        double args[] = {  };
        return iclFunction_enemy_direction.callFunction(this, args);
}

isicl.runtime.functions.MapWidth iclFunction_map_width = new isicl.runtime.functions.MapWidth();
public double _map_width() {
        double args[] = {  };
        return iclFunction_map_width.callFunction(this, args);
}

isicl.runtime.actions.Stop iclAction_stop = new isicl.runtime.actions.Stop();
public void _stop() {
        double args[] = {  };
        iclAction_stop.callAction(this, args);
}

isicl.runtime.functions.Energy iclFunction_energy = new isicl.runtime.functions.Energy();
public double _energy() {
        double args[] = {  };
        return iclFunction_energy.callFunction(this, args);
}
```

```
isicl.runtime.functions.Missed iclFunction_missed = new isicl.runtime.functions.Missed();
public double _missed() {
        double args[] = {  };
        return iclFunction_missed.callFunction(this, args);
}

isicl.runtime.actions.Fire iclAction_fire = new isicl.runtime.actions.Fire();
public void _fire(double arg01) {
        double args[] = { arg01 };
        iclAction_fire.callAction(this, args);
}

isicl.runtime.functions.HitSelf iclFunction_hit_self = new isicl.runtime.functions.HitSelf();
public double _hit_self() {
        double args[] = {  };
        return iclFunction_hit_self.callFunction(this, args);
}

isicl.runtime.functions.Random iclFunction_random = new isicl.runtime.functions.Random();
public double _random(double arg01) {
        double args[] = { arg01 };
        return iclFunction_random.callFunction(this, args);
}

isicl.runtime.functions.EnemyHeading iclFunction_enemy_heading = new isicl.runtime.functions.EnemyHeading();
public double _enemy_heading() {
        double args[] = {  };
        return iclFunction_enemy_heading.callFunction(this, args);
}

isicl.runtime.functions.Pow iclFunction_pow = new isicl.runtime.functions.Pow();
public double _pow(double arg01, double arg02) {
        double args[] = { arg01, arg02 };
        return iclFunction_pow.callFunction(this, args);
}

isicl.runtime.actions.Back iclAction_back = new isicl.runtime.actions.Back();
public void _back(double arg01) {
        double args[] = { arg01 };
        iclAction_back.callAction(this, args);
}

isicl.runtime.functions.RobotDied iclFunction_robot_died = new isicl.runtime.functions.RobotDied();
public double _robot_died() {
        double args[] = {  };
        return iclFunction_robot_died.callFunction(this, args);
}

isicl.runtime.actions.Ahead iclAction_ahead = new isicl.runtime.actions.Ahead();
public void _ahead(double arg01) {
        double args[] = { arg01 };
        iclAction_ahead.callAction(this, args);
}

isicl.runtime.functions.Tan iclFunction_tan = new isicl.runtime.functions.Tan();
public double _tan(double arg01) {
        double args[] = { arg01 };
        return iclFunction_tan.callFunction(this, args);
}

isicl.runtime.functions.Heading iclFunction_heading = new isicl.runtime.functions.Heading();
public double _heading() {
        double args[] = {  };
        return iclFunction_heading.callFunction(this, args);
}

public double _stop_moving() { return (_random((0.05))); }

isicl.runtime.functions.Speed iclFunction_speed = new isicl.runtime.functions.Speed();
public double _speed() {
        double args[] = {  };
        return iclFunction_speed.callFunction(this, args);
}

isicl.runtime.functions.EnemyDistance iclFunction_enemy_distance = new isicl.runtime.functions.EnemyDistance();
public double _enemy_distance() {
```

```
        double args[] = {  };
        return iclFunction_enemy_distance.callFunction(this, args);
}

isicl.runtime.functions.HitEnemy iclFunction_hit_enemy = new isicl.runtime.functions.HitEnemy();
public double _hit_enemy() {
        double args[] = {  };
        return iclFunction_hit_enemy.callFunction(this, args);
}

isicl.runtime.functions.Enemies iclFunction_enemies = new isicl.runtime.functions.Enemies();
public double _enemies() {
        double args[] = {  };
        return iclFunction_enemies.callFunction(this, args);
}

public double _false() { return (0); }

isicl.runtime.actions.Right iclAction_right = new isicl.runtime.actions.Right();
public void _right(double arg01) {
        double args[] = { arg01 };
        iclAction_right.callAction(this, args);
}

isicl.runtime.functions.HitBullet iclFunction_hit_bullet = new isicl.runtime.functions.HitBullet();
public double _hit_bullet() {
        double args[] = {  };
        return iclFunction_hit_bullet.callFunction(this, args);
}

isicl.runtime.functions.Xpos iclFunction_xpos = new isicl.runtime.functions.Xpos();
public double _xpos() {
        double args[] = {  };
        return iclFunction_xpos.callFunction(this, args);
}

public double _coin_toss() { return (_random((0.5))); }

isicl.runtime.functions.EnemyEnergy iclFunction_enemy_energy = new isicl.runtime.functions.EnemyEnergy();
public double _enemy_energy() {
        double args[] = {  };
        return iclFunction_enemy_energy.callFunction(this, args);
}

isicl.runtime.functions.MapHeight iclFunction_map_height = new isicl.runtime.functions.MapHeight();
public double _map_height() {
        double args[] = {  };
        return iclFunction_map_height.callFunction(this, args);
}


public void run() {
        s = new StateEngine("START");
        tv = new Vector();
        tv.add( new ConditionPair("MOVE") { public boolean condition() {
                if( (_true()) != 0.0 ) return true;
                else return false;
        } });
        tv = new Vector();
        tv.add( new ConditionPair("START") { public boolean condition() {
                if( (((0 != (_panic())) || ((_stop_moving()) != 0))? 1 : 0) != 0.0 ) return true;
                else return false;
        } });
        tv = new Vector();
        tv.add( new ConditionPair("SHOOT") { public boolean condition() {
                if( (_scanned_enemy()) != 0.0 ) return true;
                else return false;
        } });
        s.addState( new AbstractState("MOVE", tv) { public void actions() { _ahead((1)); } });


        tv = new Vector();
        tv.add( new ConditionPair("MOVE") { public boolean condition() {
                if( (_true()) != 0.0 ) return true;
                else return false;
        } });
        s.addState( new AbstractState("SHOOT", tv) { public void actions() { _fire((1)); } });
```

```
                tv = new Vector();
                tv.add( new ConditionPair("SPIN_RIGHT") { public boolean condition() {
                        if( (_true()) != 0.0 ) return true;
                        else return false;
                } });
                tv = new Vector();
                tv.add( new ConditionPair("MOVE") { public boolean condition() {
                        if( (((0 != (_panic())) || ((_stop_turning()) != 0))? 1 : 0) != 0.0 ) return true;
                        else return false;
                } });
                tv = new Vector();
                tv.add( new ConditionPair("SHOOT") { public boolean condition() {
                        if( (_scanned_enemy()) != 0.0 ) return true;
                        else return false;
                } });
                s.addState( new AbstractState("SPIN_RIGHT", tv) { public void actions() { _right((1)); } });


                tv = new Vector();
                tv.add( new ConditionPair("SPIN_LEFT") { public boolean condition() {
                        if( (_true()) != 0.0 ) return true;
                        else return false;
                } });
                tv = new Vector();
                tv.add( new ConditionPair("MOVE") { public boolean condition() {
                        if( (((0 != (_panic())) || ((_stop_turning()) != 0))? 1 : 0) != 0.0 ) return true;
                        else return false;
                } });
                tv = new Vector();
                tv.add( new ConditionPair("SHOOT") { public boolean condition() {
                        if( (_scanned_enemy()) != 0.0 ) return true;
                        else return false;
                } });
                s.addState( new AbstractState("SPIN_LEFT", tv) { public void actions() { _left((1)); } });


                tv = new Vector();
                tv.add( new ConditionPair("SPIN_RIGHT") { public boolean condition() {
                        if( (_true()) != 0.0 ) return true;
                        else return false;
                } });
                tv = new Vector();
                tv.add( new ConditionPair("SPIN_LEFT") { public boolean condition() {
                        if( (_coin_toss()) != 0.0 ) return true;
                        else return false;
                } });
                s.addState( new AbstractState("START", tv) { public void actions() { _pass(); } });



                while(true) {s.tick();}
        }
}
```

Chapter 7

Lessons Learned

7.1 Michele

Things always take longer than you expect, even if you expect them to take longer than you want them to. Firm deadlines help keep the project on task and moving along. Everyone needs to know exactly what you're trying to accomplish and why and they all need to agree. It's hard to set deadlines for someone else when everyone has the same status. Detailed understanding of the project and detailed plans are important to keeping the team members engaged, focused and motivated. People with diverse backgrounds, age and gender differences can have conflicting ideas and viewpoints on what is important. Doing things at the last minute isn't fun. Make sure group information and knowledge is shared so if you lose a member of your group, you don't have an information void as well.

7.2 Matthew

I can't honestly say that I "learned" that much from this group project experience, since a lot of the things that went wrong have gone wrong before, and probably will again. It did reinforce a few things, though. Communicate early and often, but not by forcing a meeting when you don't have anything to talk about. Fix project requirements and roles as early as possible. Make the design as simple as possible, but flexible enough to accommodate future expansion. (That's one thing that we did pretty well.) Finally, share work and notes with the group often, in case unforseen circumstances cause a team member to be unable to fulfill his role.

7.3 Michael

First and foremost I learned just how much work goes into a compiler, even for a language as small in scope as ours. Advance planning is key, especially when it comes to having a clear idea of what a language should do. I learned to pick my battles when my ideas clashed with those of other group members, and I realized how important compromise can be to keeping things moving in those situations. I also picked up a great deal of practical skills: using obscure corners of the Java 1.4 API, document layout with LaTeX, and especially how to work effectively on a group software project.

Appendix A

Source code

A.1 Grammar

A.1.1 ISICLLexer.g

```
header {
package isicl;

/**
 * The ISICL Lexer. Feeds ISICLParser.
 *
 * @author Matthew Keitz
 */
}

class ISICLLexer extends Lexer;

options {
    exportVocab = ISICL;
    charVocabulary = '\003'..'\377';
        k = 2;
        testLiterals = false;
    caseSensitiveLiterals = false;
}
tokens {
    KEYWORD_STATE = "state";
    KEYWORD_COND = "condition";
    KEYWORD_CONST = "const";
    KEYWORD_FUNC = "function";
    KEYWORD_ACTION = "action";
    OP_AND = "and";
    OP_OR = "or";
    OP_NOT = "not";
    OP_XOR = "xor";
}

protected
NEWLINE     : ("\r\n" | '\n' | '\r') {newline();};

WS
    :
        (' ' | '\t' | '\f' | NEWLINE)
        {$setType(Token.SKIP);}
    ;

COMMENT
    :
        "//" (~('\n' | '\r'))* (NEWLINE)?
                {$setType(Token.SKIP);}
    ;
```

```
LPAREN      : '(';
RPAREN          : ')';
LBRACK          : '[';
RBRACK          : ']';
ARROW           : "->";
PLUS            : '+';
MINUS           : '-';
TIMES           : '*';
DIV                 : '/';
EQ              : '=';
GT                  : '>';
LT              : '<';
GEQ             : ">=";
LEQ                 : "<=";
NEQ             : "!=";
COLON           : ':';
COMMA           : ',';

protected
DOT         : '.';

protected
DIGIT           : ('0'..'9');

protected
ALPHA           : ('a'..'z') | ('A'..'Z');

// Identifiers, including "qualified identifiers",
// i.e. Java classes in packages.
ID
options {
    testLiterals = true;
    paraphrase = "an identifier";
}
    :
        (ALPHA | '_' | '$')
        (ALPHA | '_' | '$' | DIGIT)*
    ;

// I can't figure out why ANTLR has a problem with this
// rule. It seems to think that ".e" or ".E" could be
// part of a number, but the FRAC rule clearly says
// that a decimal point cannot appear without at least
// one digit following it. I'm almost certain that this
// is an ANTLR bug. In fact, even turning off the
// warning doesn't seem to work.
QUAL_ID
options {paraphrase = "a qualified identifier";}
        :
        (options {generateAmbigWarnings = false;} : DOT ID)
    ;

// Floating point literals.
LITERAL
options {paraphrase = "a number";}
        :
        (options {generateAmbigWarnings = false;} :
            (
                (INT (FRAC)? EXPONENT)
            |
                (FRAC EXPONENT)
            )
        )
        {
            // This should catch syntactically valid literals
            // that are just too long for the Java compiler
            // to accept.
            if (Double.valueOf(getText()).isInfinite())
                throw new SemanticException("Number out of range.", getFilename(), getLine(), getColumn());
        }
    ;

protected
INT         : (DIGIT)+;

protected
```

```
EXPONENT        : (('E' | 'e') ('+' | '-')? (DIGIT)+)?;

protected
FRAC        : DOT INT;
```

A.1.2 ISICLParser.g

```
header {
package isicl;

/**
 * The ISICL Parser. Eventually generates an AST from
 * an ISICL input stream. Checks all syntax errors.
 *
 * @author Matthew Keitz
 */
}

class ISICLParser extends Parser;

options
{
    importVocab = ISICL;
    buildAST = true;

    // ExtendedAST tracks line and column numbers
    // (for some strange reason, ANTLR's own AST
    // doesnt'.)
    ASTLabelType = "ExtendedAST";
}
tokens {UNARY_MINUS;}

robot             : (declaration)* EOF!;

declaration       : state_decl | cond_decl | const_decl | func_decl | action_decl;

state_decl
    :
        KEYWORD_STATE^ ID COLON!
        action_call
        LBRACK! (transition)* RBRACK!
    ;

cond_decl
    :
        KEYWORD_COND^ ID COLON! expr
    ;

const_decl
    :
        KEYWORD_CONST^ ID COLON! LITERAL
    ;

func_decl
    :
        KEYWORD_FUNC^ ID COLON! java_class
    ;

action_decl
    :
        KEYWORD_ACTION^ ID COLON! java_class
    ;

java_class  : ID (QUAL_ID)*;

action_call : ID^ (expr (COMMA! expr)*)?;

transition  : expr ARROW^ ID;

expr        : not_expr ((OP_AND^ | OP_OR^ | OP_XOR^) not_expr)*;

not_expr    : (OP_NOT^ comp_expr) | comp_expr;

comp_expr   : num_expr ((EQ^ | GT^ | LT^ | GEQ^ | LEQ^ | NEQ^) num_expr)*;

num_expr    : term_expr ((PLUS^ | MINUS^) term_expr)*;
```

```
term_expr   : unary_expr ((TIMES^ | DIV^) unary_expr)*;

unary_expr
    :
        (PLUS!)? value_expr
    |
        minus:MINUS^ value_expr {minus.setType(UNARY_MINUS);}
    ;

value_expr        :
        LITERAL
    |
        func_expr
    |
        LPAREN! expr RPAREN!
    ;

func_expr : ID^ (LPAREN! (expr (COMMA! expr)*)? RPAREN!)?;
```

A.1.3 Compiler.g

```
header {
    package isicl;

/**
 * The ISICL Compiler.
 * Manages the ISICL Lexer and Parser, handles semantic
 * checking, and generates an output file. Includes
 * generateCode method, which belongs to Michael Marcus.
 *
 * @author Matthew Keitz
 */

    import java.util.*;
    import java.io.*;
    import java.net.*;
}

class Compiler extends TreeParser;

options {
    ASTLabelType = "ExtendedAST";
    importVocab = ISICL;
}

{
    private SymbolTable symbols;

    /**
     * Reads from the source file, and writes a Java class to the
     * output stream.
     *
     * @param source name of a source code file to compile
     * @param robotName name of the robot class generated
     * @param headers List of file name Strings, each of which
     * will be loaded before the input file
     * @param classPath List of path Strings, which will be
     * appended to the existing class path when loading proc
     * implementations
     * @param out output stream
     * @param err error stream
     * @param haltOnErrors whether output will not be written if
     * there are semantic errors
     * @return false if there were any errors; true otherwise.
     * @throws FileNotFoundException if any of the input files
     * could not be found
     * @throws MalformedURLException if a class path entry is
     * not valid
     * @throws IOException if there is a problem generating
     * output
     */
    public boolean compile(String source, String robotName, List headers, List classPath, PrintStream out, PrintStream err, boolean haltOnErrors)
        throws FileNotFoundException, MalformedURLException, IOException
        {
            URL[] classURLs = new URL[classPath.size()];
```

```
        for (int i = 0; i < classPath.size(); i++)
            classURLs[i] = (URL)classPath.get(i);

        Loc.reset();

        symbols = new SymbolTable(classURLs);

        boolean errors = false;
        for (Iterator i = headers.iterator(); i.hasNext();)
            errors |= parse((String)i.next(), err);

        parse(source, err);

        errors |= !symbols.resolveSymbols(err);

        if (errors && haltOnErrors)
            return false;

        generateCode(out, robotName);

        return !errors;
    }

/**
 * Parses a single file, without resetting the compiler's state.
 *
 * @param file name of a source file
 * @param err error stream
 * @return true if there are syntax errors
 * @throws FileNotFoundException if the source file cannot be
 * found
 * @throws IOException if there is a problem writing to the
 * error stream
 */
private boolean parse(String file, PrintStream err)
    throws FileNotFoundException, IOException
    {
        Loc.setFilename(file);
        FileInputStream in = new FileInputStream(file);

        ISICLLexer lexer = new ISICLLexer(in);
        ISICLParser parser = new ISICLParser(lexer);

        parser.setASTNodeClass("isicl.ExtendedAST");

        try {
            parser.robot();
            robot(parser.getAST());
        }
        catch (ANTLRException e) {
            // As far as I can tell, this exception
            // occurs only if there is a syntax error
            // at the top level of the source file.

            err.println(e);
            in.close();
            return true;
        }

        in.close();
        return false;
    }

/**
 * Outputs a Robot class, using the objects in the symbol table.
 * This method is based on code originally written for Main.java
 * by Michael Marcus.
 *
 * @param out output stream
 * @param robotName name of the robot class generated
 */
private void generateCode(PrintStream out, String robotName) {
    out.println("import isicl.runtime.*;");
    out.println("import java.util.*;");
    out.println();
```

```
        // Define robot class
        out.println( "public class " + robotName + " extends ISICLRobot {");
        out.println();

        // Instantiate user-defined actions and functions
        for (Enumeration procs = symbols.getProcs(); procs.hasMoreElements();)
            out.println( "\t" + ((Proc)procs.nextElement()).getCode() );
        out.println();

        out.println("\tpublic void run() {");

        // Initialize the state engine with the initial state
        out.println("\t\ts = new StateEngine(\"START\");");

        // Add states to the state engine

        // Iterate through the symbol table for states and generate code for each one.
        for (Enumeration states = symbols.getStates(); states.hasMoreElements();)
            out.println( "\t\t" + ((State)states.nextElement()).getCode() );
        out.println();

        // Start the state engine
        out.println("\t\twhile(true) {s.tick();}");

        // Tie up loose curly brackets
        out.println("\t}");
        out.println("}");
    }
}

robot
    :
        (state_decl | func_decl | action_decl | cond_decl | const_decl)*
    ;

// Returns a Java expression that evaluates to a boolean.
// This conversion stuff could be a whole lot prettier.
bool_expr
returns [String java_expr = null]
{String x, y;}
    :
        (
            #(OP_AND x=double_expr y=double_expr) {java_expr = "((0 != " + x + ") && (" + y + " != 0))";}
        |
            #(OP_OR x=double_expr y=double_expr) {java_expr = "((0 != " + x + ") || (" + y + " != 0))";}
        |
            #(OP_XOR x=double_expr y=double_expr) {java_expr = "((0 != " + x + ") ^ (" + y + " != 0))";}
        |
            #(OP_NOT x=double_expr) {java_expr = "(" + x + " == 0)";}
        )
    ;

// Returns a Java expression that evaluates to a double.
double_expr
returns [String java_expr = null]
{String x, y;}
    :
        (
            x=bool_expr {java_expr = x + "? 1 : 0";}
        |
            #(EQ x=double_expr y=double_expr) {java_expr = x + " == " + y;}
        |
            #(GT x=double_expr y=double_expr) {java_expr = x + " > " + y;}
        |
            #(LT x=double_expr y=double_expr) {java_expr = x + " < " + y;}
        |
            #(GEQ x=double_expr y=double_expr) {java_expr = x + " >= " + y;}
        |
            #(LEQ x=double_expr y=double_expr) {java_expr = x + " <= " + y;}
        |
            #(NEQ x=double_expr y=double_expr) {java_expr = x + " != " + y;}
        |
            #(UNARY_MINUS x=double_expr) {java_expr = " -" + x;}
        |
            #(PLUS x=double_expr y=double_expr) {java_expr = x + " + " + y;}
        |
            #(MINUS x=double_expr y=double_expr) {java_expr = x + " - " + y;}
```

```
        |
            #(TIMES x=double_expr y=double_expr) {java_expr = x + " * " + y;}
        |
            #(DIV x=double_expr y=double_expr) {java_expr = x + " / " + y;}
        |
            java_expr=func_expr
        |
            num:LITERAL {
                // We MUST check for errors here.
                java_expr = num.getText();
            }
        )
        {java_expr = "(" + java_expr + ")";}
    ;

// Returns a Java class name (possibly qualified.)
java_class
returns [String name = null]
    :
        first:ID {name = first.getText();}
        (rest:QUAL_ID {name += rest.getText();})*
    ;

// Returns a function call.
func_expr
returns [String java_expr = null]
{
    int count = 0;
    String arg;
}
    :
        #(name:ID {java_expr = "_" + name.getText().toLowerCase() + "(";}
            (arg=double_expr {java_expr += arg; count++;}
                (arg=double_expr {java_expr += ", " + arg; count++;})*
            )?
            {java_expr += ")";}
        )
        // Note function reference.
        {symbols.referenceProc(new Proc.ProcRef(name.getText(), count, Proc.ProcType.FUNCTION, name.getLoc()));}
    ;

cond_decl
{
    UserFunction func;
    String definition;
}
    :
        #(
            KEYWORD_COND
            name:ID {func = new UserFunction(name.getText(), name.getLoc());}
            definition=double_expr {func.setDefinition(definition);}
        )
        {symbols.defineProc(func);}
    ;

const_decl
{
    UserFunction func;
    String definition;
}
    :
        #(
            KEYWORD_CONST
            name:ID {func = new UserFunction(name.getText(), name.getLoc());}
            definition=double_expr {func.setDefinition(definition);}
        )
        {symbols.defineProc(func);}
    ;

func_decl
{
    Function func;
    String class_name;
}
    :
        #(KEYWORD_FUNC name:ID class_name=java_class) {
            func = new Function(name.getText(), name.getLoc());
```

```
            func.setClass(class_name);
            symbols.defineProc(func);
        }
    ;

action_decl
{
    Action action;
    String class_name;
}
    :
        #(KEYWORD_ACTION name:ID class_name=java_class) {
            action = new Action(name.getText(), name.getLoc());
            action.setClass(class_name);
            symbols.defineProc(action);
        }
    ;

state_decl
{
    State state;
    String arg, condition;
    int count = 0;
}
    :
        #(KEYWORD_STATE
            name:ID {state = new State(name.getText(), name.getLoc());}
            #(
                action:ID {state.setAction(action.getText());}
                (arg=double_expr {state.addArg(arg); count++;})*
                {symbols.referenceProc(new Proc.ProcRef(action.getText(), count, Proc.ProcType.ACTION, action.getLoc()));}
            )
            (
                #(
                    ARROW condition=double_expr new_state:ID
                    {
                        state.addTransition(condition, new_state.getText());
                        symbols.referenceState(new State.StateRef(new_state.getText(), new_state.getLoc()));
                    }
                )
            )*
        )
        {symbols.defineState(state);}
    ;
```

A.2 Compiler

A.2.1 isicl/Main.java

```
package isicl;

import java.io.*;
import java.util.*;
import java.lang.*;

/**
 * <p>The entry point for the ISICL compiler.</p>
 * <p>(Write some more stuff about usage and flags.)</p>
 *
 * @author Michele Cozart
 * @version $Id: Main.java,v 1.15 2003/05/14 03:55:27 mlm211 Exp $
 * @see isicl
 */
class Main {

    public static void main( String[] args )
    {
        Vector headers = new Vector();
        headers.add("./isicl/standard.icl");

        String filename = args[0];

        try
```

```
        {
           if( (filename.length() > 4) && filename.substring(filename.length()-4).equals(".icl") )
           {
               boolean compileSuccess = false;
               boolean returnValue = false;

               String filenameNoExt = filename.substring( filename.lastIndexOf('/')+1, filename.length()-4 );
               filenameNoExt = filenameNoExt.substring(0,1).toUpperCase()
                   + filenameNoExt.substring(1,filenameNoExt.length());

               PrintStream out = new PrintStream(new FileOutputStream(filenameNoExt + ".java"));

               Compiler compiler = new Compiler();


               compileSuccess = compiler.compile(filename, filenameNoExt, headers, new Vector(0), out, System.err, true);

               if (compileSuccess)
               {
                  System.out.println("Compilation successful.");
               }
               else
               {
                   System.out.println("Compilation failed!");
               }
           }
           else
           {
               System.err.println("Usage: java isicl.Main <directory or file name>");
           }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

A.2.2 isicl/SymbolTable.java

```
package isicl;

/**
 * It's a symbol table.
 *
 * @author  Matthew Keitz
 * @version $Id: SymbolTable.java,v 1.10 2003/05/13 19:29:42 msk2102 Exp $
 */

import java.util.*;
import java.io.*;
import java.net.*;

class SymbolTable {
    private URLClassLoader classes = new URLClassLoader(new URL[0]);

    private TreeSet errors = new TreeSet();

    private Hashtable states = new Hashtable();
    private Hashtable procs = new Hashtable();

    private LinkedList stateRefs = new LinkedList();
    private LinkedList procRefs = new LinkedList();

    /**
     * Initializes a new symbol table.
     *
     * @param classPath class path to use when searching for proc
     * implementations (in addition to the usual search path.)
     */
    public SymbolTable(URL[] classPath) {
        classes = new URLClassLoader(classPath);

        // This dummy reference causes an error if there is no
        // START state.
```

```
    stateRefs.add(new State.StateRef("START", Loc.system()));
}

/**
 * Adds a state definition to the table.
 *
 * @param state the new definition
 */
public void defineState(State state) {
    State old = (State)states.put(state.getName(), state);
    if (old != null)
        errors.add(state.redefines(old));
}

/**
 * Records a new reference to a state.
 *
 * @param state the state reference
 */
public void referenceState(State.StateRef state)
{stateRefs.add(state);}

/**
 * Adds a new proc definition to the table.
 *
 * @param proc the new definition
 */
public void defineProc(Proc proc) {
    Proc old = (Proc)procs.put(proc.getName(), proc);
    if (old != null)
        errors.add(proc.redefines(old));
}

/**
 * Records a new reference to a proc.
 *
 * @param proc the proc reference
 */
public void referenceProc(Proc.ProcRef proc)
{procRefs.add(proc);}

/**
 * Resolves symbols, and outputs all semantic errors.
 *
 * @param out a stream to use for error output
 * @return true if there are errors, false otherwise
 */
public boolean resolveSymbols(PrintStream out) {

    // Check implementations.
    for (Iterator i = procs.values().iterator(); i.hasNext();) {
        Error err = ((Proc)i.next()).checkImplementation(classes);

        if (err != null) {
            i.remove();
            errors.add(err);
        }
    }


    // Check state references.
    for (Iterator i = stateRefs.iterator(); i.hasNext();) {
        State.StateRef ref = (State.StateRef)i.next();
        Error err = ref.check((State)states.get(ref.getName()));

        if (err != null)
            errors.add(err);
    }

    // Check proc references.
    for (Iterator i = procRefs.iterator(); i.hasNext();) {
        Proc.ProcRef ref = (Proc.ProcRef)i.next();
        Error err = ref.check((Proc)procs.get(ref.getName()));

        if (err != null)
            errors.add(err);
    }
```

```
        // Output errors.
        for (Iterator i = errors.iterator(); i.hasNext(); out.println((Error)i.next()));

        return errors.isEmpty();
    }

    /**
     * Gets the set of state definitions.
     *
     * @return an enumeration for states
     */
    public Enumeration getStates()
    {return states.elements();}

    /**
     * Gets the set of proc definitions.
     *
     * @return an enumeration for procs
     */
    public Enumeration getProcs()
    {return procs.elements();}
}
```

A.2.3 isicl/State.java

```
package isicl;

import java.util.*;
import java.io.*;

/**
 * <p>Description: This class implements a state.  Conditions and newStates come in
 * pairs, representing the transitions.<p>
 *
 * @author  Michele Cozart
 * @author  Matthew Keitz
 * @author  Michael Marcus
 * @version $Id: State.java,v 1.15 2003/05/13 22:02:37 mlm211 Exp $
 * <p>Project:  ISICL  COMS4115</p>
 */
class State
{
    /**
     * Description: This class implements a state reference.
     *
     * @author   Matt Keitz
     * @version  $Id: State.java,v 1.15 2003/05/13 22:02:37 mlm211 Exp $
     * <p>Project:  ISICL  COMS4115</p>
     */
    public static class StateRef
    {
        private String name;
        private Loc loc;

        /**
         * Description: StateRef constructor
         *
         * @param name the state name
         * @param line line number of reference
         * @param column column number of reference
         * Creation date: April 2003
         */
        public StateRef(String name, Loc loc) {
            this.name = name.toUpperCase();
            this.loc = loc;
        }

        /**
         * Description: Returns the name of a state.
         *
         * @return the name of the state
         * Creation date: April 2003
         */
        public String getName()
```

```
    {return name;}

    /**
     * Verifies that this reference is consistent with the
     * state definition. (Right now, the only way for the
     * reference to be inconsistent is if the state wasn't
     * defined at all.
     *
     * @param state the state whose reference needs to be checked
     * @return an error object, or null if there is no error.
     * @see State
     * Creation date: April 2003
     */
    public Error check(State state) {
        if (state == null)
            return Error.stateNotDefined(name, loc);
        else
            return null;
    }
}

private String name, action;
private Loc loc;
private Stack args = new Stack();
private Stack conditions = new Stack();
private Stack newStates = new Stack();

/**
 * Description: Constructs a state object.
 *
 * @param name the name of the state
 * @param line line in which state was defined
 * @param column column in which state was defined
 * Creation date: April 2003
 */
public State(String name, Loc loc) {
    this.name = name.toUpperCase();
    this.loc = loc;
}

/**
 * Description: Sets a state action
 *
 * @param action the action to be performed while in the state
 */
public void setAction(String action)
{this.action = action.toLowerCase();}

/**
 * Description: Adds an argument to the the arg stack (for the action)
 *
 * @param arg the argument to add
 */
public void addArg(String arg)
{args.push(arg);}

/**
 * Description: Returns the number of args currently on the arg stack
 *
 * @return the number of arguments to the state's action
 */

// Is this method necessary? Why?
//public int argCount()
//{
//return args.size();
//}

/**
 * Description: Returns the name of a state.
 *
 * @return the state name as a String
 */
public String getName()
{return name;}

public Error redefines(State state)
```

```
    {return Error.stateRedefined(name, loc, state.loc);}

    /**
     * Description: Adds a transition (condition, newState) to the conditions and
     * newStates stacks.
     * Error handling still needs to be added.
     *
     * @param condition the condition as a String containing a Java expression
     * @param newState  a String representing the name of the destination state
     */
    public void addTransition(String condition, String newState) {
        newStates.push(newState.toUpperCase());
        conditions.push(condition);
    }

    /**
     * Description: Generates Java code for this State object.
     *
     * @return a String containing valid Java code that implements the state
     */
    public String getCode()
    {
        // Initialize a new string buffer
        StringBuffer code = new StringBuffer(1024);

        // Generate code for each condition-transition pair
        while( !conditions.empty() || !newStates.empty() ) { // just to be safe
            code.append( "tv = new Vector();\n\t\t" );
            code.append( "tv.add( new ConditionPair(\"" ).append( (String)newStates.pop() ).append( "\") " );

            // Conditions - call the method "_<condition>()"
            code.append( "{ public boolean condition() {\n\t\t" );
            code.append( "\tif( " ).append( (String)conditions.pop() ).append( " != 0.0 ) return true;\n\t\t" );
            code.append( "\telse return false;\n\t\t" );
            code.append( "} });\n\t\t" );
        }

        // Generate code to create the state
        code.append( "s.addState( new AbstractState(\"" ).append( name ).append( "\", tv) ");

        // Generate call to action method
        code.append( "{ public void actions() { _" ).append( action ).append( "(" );
        while( !args.empty() ) {
            code.append( (String)args.pop() );
            if( !args.empty() ) {
                code.append( ", " );
            }
        }
        code.append( ");" );

        // Clean up
        code.append( " } });\n\n" );

        // Return the code string
        return code.toString();
    }
}
```

A.2.4 isicl/Proc.java

```
package isicl;

/**
 * Interface for all ISICL functions and actions.
 *
 * @author  Matthew Keitz
 * @version $Id: Proc.java,v 1.8 2003/05/13 04:13:17 msk2102 Exp $
 */
abstract class Proc {

    /**
     * Name of proc (in ISICL).
     */
    protected String name;
```

```
/**
 * Number of arguments expected.
 */
protected int numArgs;

/**
 * Location of declaration.
 */
protected Loc loc;

/**
 * Enum class for actions vs. functions.
 */
public static class ProcType {
    private final String name, nameUpper, nameArticle;

    private ProcType(String name, String nameUpper, String nameArticle) {
        this.name = name;
        this.nameUpper = nameUpper;
        this.nameArticle = nameArticle;
    }

    // The following methods are used when
    // generating messages.

    /**
     * Name of type.
     *
     * @return type name
     */
    public String toString()
    {return name;}

    /**
     * Proper-case version of type name.
     *
     * @return type name in proper case
     */
    public String toStringUpper()
    {return nameUpper;}

    /**
     * Type name with indefinite article.
     *
     * @return type name with indefinite article.
     */
    public String toStringArticle()
    {return nameArticle;}

    /**
     * Constant representing actions.
     */
    public static final ProcType ACTION = new ProcType("action", "Action", "an action");

    /**
     * Constant representing functions.
     */
    public static final ProcType FUNCTION = new ProcType("function", "Function", "a function");
}

/**
 * This class represents a reference to a proc.
 */
public static class ProcRef {
    private String name;
    private int numArgs;
    private Loc loc;
    private ProcType type;

    /**
     * Constructs a new reference.
     *
     * @param name name of proc referenced
     * @param numArgs number of arguments used in reference
     * @param type context of reference
     * @param loc location of reference
     */
```

```
    public ProcRef(String name, int numArgs, ProcType type, Loc loc) {
        this.name = name.toLowerCase();
        this.numArgs = numArgs;
        this.type = type;
        this.loc = loc;
    }

    /**
     * Gets the name of the referenced proc.
     *
     * @return proc name
     */
    public String getName()
    {return name;}

    /**
     * Verifies that this reference is compatible
     * with the proc definition given.
     *
     * @param proc a proc definition
     * @return an Error object, if appropriate, or null
     */
    public Error check(Proc proc) {
        if (proc == null)
            return Error.procNotDefined(name, loc, type);
        else if (proc.getType() != type)
            return Error.procContextInconsistent(name, type, loc, proc.getType(), proc.loc);
        else if (proc.numArgs != numArgs)
            return Error.procWrongNumberOfArgs(name, type, numArgs, loc, proc.numArgs, proc.loc);

        return null;
    }
}

/**
 * Constructs a new proc definition (as part of a derived class.)
 *
 * @param name name of proc being defined
 * @param loc location of declaration
 */
public Proc(String name, Loc loc) {
    this.name = name;
    this.loc = loc;
}

/**
 * Verifies that there is a working implementation for this proc.
 *
 * @param classes loader to use when searching for implementations
 * @return an Error object, if appropriate, or null.
 */
public abstract Error checkImplementation(ClassLoader classes);

/**
 * Gets the name of the proc defined.
 *
 * @return proc name
 */
public String getName()
{return name;}

/**
 * Generates an error indicating that the proc has been redefined.
 * This is a member of Proc to simply accessibility issues.
 *
 * @param proc the original proc definition
 * @return an error object
 */
public Error redefines(Proc proc)
{return Error.procRedefined(name, loc, getType(), proc.loc, proc.getType());}

/**
 * Generates code implementing this proc.
 *
 * @return Java code
 */
public abstract String getCode();
```

```
    /**
     * Gets the type of this proc.
     *
     * @return proc type
     */
    public abstract ProcType getType();
}
```

A.2.5 isicl/Function.java

```
package isicl;

/**
 * Contains the definition of an ISICLfunaction.
 * The getCode method belongs to Michael Marcus.
 *
 * @author  Matthew Keitz
 * @version $Id: Function.java,v 1.16 2003/05/14 00:55:31 msk2102 Exp $
 */

import java.util.*;

class Function extends Proc
{
    private String className;

    /**
     * Constructs a new function definition.
     *
     * @param name name of the function being defined
     * @param loc location of declaration
     */
    public Function(String name, Loc loc)
    {super(name, loc);}

    /**
     * Sets the class that implements this function.
     *
     * @param name name of the implementing class
     */
    public void setClass(String className)
    {this.className = className;}

    /**
     * Gets the type of this proc.
     *
     * @return ACTION
     */
    public Proc.ProcType getType()
    {return Proc.ProcType.FUNCTION;}

    /**
     * Verifies that there is a working implementation for this function.
     *
     * @param classes loader to use when searching for implementations
     * @return an Error object, if appropriate, or null.
     */
    public Error checkImplementation(ClassLoader classes) {
        Object impl;
        try {impl = Class.forName(className, true, classes).newInstance();}
        catch (Exception e) {return Error.implementationNotFound(loc, className);}

        isicl.interfaces.FunctionImplementation function;
        try {function = (isicl.interfaces.FunctionImplementation)impl;}
        catch (Exception e) {return Error.badImplementation(loc, className, Proc.ProcType.FUNCTION);}

        // If we made it here, we have a good implementation.
        numArgs = function.getNumArgs();

        return null;
    }

    /**
     * Description: Generates Java code for this Function object.
```

```
     * This method belongs to Michael Marcus.
     *
     * @return a String containing valid Java code that implements the function
     */
    public String getCode()
    {
        StringBuffer code = new StringBuffer();

        // Instantiate the function
        code.append( className ).append( " iclFunction_" ).append( name );
        code.append( " = new " ).append( className ).append( "();\n\t" );

        // Generate wrapper method
        code.append( "public double _" ).append( name ).append( "(" );
        for( int arg = 1; arg <= numArgs; arg++ ) {
            if( arg < 10 )
                code.append( "double arg0" ).append( arg );
            else
                code.append( "double arg"  ).append( arg );
            if( arg < numArgs )
                code.append( ", " );
        }
        code.append( ") {\n\t\t" );
        code.append( "double args[] = { " );
        for( int arg = 1; arg <= numArgs; arg++ ) {
            if( arg < 10 )
                code.append( "arg0" ).append( arg );
            else
                code.append( "arg"  ).append( arg );
            if( arg < numArgs )
                code.append( ", " );
        }
        code.append( " };\n\t\t" );
        code.append( "return iclFunction_" ).append( name ).append( ".callFunction(this, args);\n\t}\n" );

        return code.toString();
    }
}
```

A.2.6 isicl/UserFunction.java

```
package isicl;

/**
 * This class is for functions defined in ISICL (conditions and consts).
 * These functions always have zero args.
 * The code generated simply evaluates the expression in definition.
 * Includes getCode method, which belongs to Michael Marcus.
 *
 * @author Matthew Keitz
 * @author Michael Marcus
 * @version $Id: UserFunction.java,v 1.9 2003/05/13 04:13:17 msk2102 Exp $
 * @see Proc
 */

import java.io.*;
import java.util.StringTokenizer;
import java.util.*;

class UserFunction extends Proc
{
    private String definition;

    /**
     * Creates a new UserFunction, which always has zero arguments.
     *
     * @param name function name
     * @param loc location of definition
     */
    public UserFunction(String name, Loc loc) {
        super(name, loc);
        numArgs = 0;
    }

    /**
```

```
     * Sets the definition expression for a function.
     *
     * @param definition an expression, in Java
     */
    public void setDefinition(String definition)
    {this.definition = definition;}

    /**
     * Gets the type of this proc.
     *
     * @return FUNCTION
     */
    public Proc.ProcType getType()
    {return Proc.ProcType.FUNCTION;}

    /**
     * Verifies that there is a working implementation for this proc.
     * For UserFunctions, the implementation is always valid.
     *
     * @param classes loader to use when searching for implementations
     * @return null (that is, no Error)
     */
    public Error checkImplementation(ClassLoader classes)
    {return null;}

    /**
     * Description: Generates Java code for this UserFunction object.
     * This method belongs to Michael Marcus.
     *
     * @return a String containing valid Java code that implements the
     * specified condition or constant
     */
    public String getCode()
    {
        StringBuffer code = new StringBuffer();

        // Generate wrapper method
        code.append( "public double _" ).append( name ).append( "() { return " );
        code.append( definition );
        code.append( "; }\n" );

        return code.toString();
    }
}
```

A.2.7 isicl/Action.java

```
package isicl;

/**
 * Contains the definition of an ISICL action.
 * The getCode method belongs to Michael Marcus.
 *
 * @author  Matthew Keitz
 * @version $Id: Action.java,v 1.15 2003/05/14 00:55:31 msk2102 Exp $
 */

import java.util.*;

class Action extends Proc
{
    private String className;

    /**
     * Constructs a new action definition.
     *
     * @param name name of the action being defined
     * @param loc location of declaration
     */
    public Action(String name, Loc loc)
    {super(name, loc);}

    /**
     * Sets the class that implements this action.
     *
```

```
     * @param name name of the implementing class
     */
    public void setClass(String className)
    {this.className = className;}

    /**
     * Gets the type of this proc.
     *
     * @return ACTION
     */
    public Proc.ProcType getType()
    {return Proc.ProcType.ACTION;}

    /**
     * Verifies that there is a working implementation for this action.
     *
     * @param classes loader to use when searching for implementations
     * @return an Error object, if appropriate, or null.
     */
   public Error checkImplementation(ClassLoader classes) {
        Object impl;
        try {impl = Class.forName(className, true, classes).newInstance();}
        catch (Exception e) {return Error.implementationNotFound(loc, className);}

        isicl.interfaces.ActionImplementation action;
        try {action = (isicl.interfaces.ActionImplementation)impl;}
        catch (Exception e) {return Error.badImplementation(loc, className, Proc.ProcType.ACTION);}

        // If we made it here, we have a good implementation.
        numArgs = action.getNumArgs();

        return null;
    }

    /**
     * Description: Generates Java code that declares this Action object as an instance
     * of its class.
     *
     * @return a String containing valid Java code that implements the action
     */
    public String getCode()
    {
        StringBuffer code = new StringBuffer();

        // Instantiate the action
        code.append( className ).append( " iclAction_" ).append( name );
        code.append( " = new " ).append( className ).append( "();\n\t" );

        // Generate wrapper method
        code.append( "public void _" ).append( name ).append( "(" );
        for( int arg = 1; arg <= numArgs; arg++ ) {
            if( arg < 10 )
                code.append( "double arg0" ).append( arg );
            else
                code.append( "double arg"  ).append( arg );
            if( arg < numArgs )
                code.append( ", " );
        }
        code.append( ") {\n\t\t" );
        code.append( "double args[] = { " );
        for( int arg = 1; arg <= numArgs; arg++ ) {
            if( arg < 10 )
                code.append( "arg0" ).append( arg );
            else
                code.append( "arg"  ).append( arg );
            if( arg < numArgs )
                code.append( ", " );
        }
        code.append( " };\n\t\t" );
        code.append( "iclAction_" ).append( name ).append( ".callAction(this, args);\n\t}\n\t" );

        return code.toString();
    }
}
```

A.2.8 isicl/Error.java

```
package isicl;

/**
 * Contains generators for all semantic error messages.
 * Keeping them in one place makes them easier to manage.
 *
 * @author  Matt Keitz
 * @version $Id: Error.java,v 1.5 2003/05/14 00:55:31 msk2102 Exp $
 */
class Error implements Comparable {
    private Loc loc;
    private String text;

    /**
     * Constructs a new error, with the given message.
     *
     * @param loc location of error
     * @param text error message
     */
    public Error(Loc loc, String text) {
        this.loc = loc;
        this.text = text;
    }

    /**
     * Defines an order between Error objects.
     * This is necessary because errors may not be
     * generated in the order that they appear in
     * the source file(s).
     *
     * @param x another Error object
     * @return the order of each Error's location or text,
     * or 0 if both are equal.
     */
    public int compareTo(Object x) {
        int result = loc.compareTo(((Error)x).loc);
        if (result == 0)
            result = text.compareTo(((Error)x).text);

        return result;
    }

    public String toString()
    {return loc + ": " + text;}

    /**
     * Generated when a state is redefined.
     *
     * @param name name of the state
     * @param current location of second declaration
     * @param old location of first definition
     * @return an Error object
     */
    public static Error stateRedefined(String name, Loc current, Loc old)
    {return new Error(current, "State " + name + " was originally defined at " + old + ".");}

    /**
     * Generated when a proc is redefined.
     *
     * @param name name of proc
     * @param current location of second declaration
     * @param type type of second declaration
     * @param old location of first declaration
     * @param oldType type of first declaration
     * @return an Error object
     */
    public static Error procRedefined(String name, Loc current, Proc.ProcType type, Loc old, Proc.ProcType oldType) {
        if (type == oldType)
            return new Error(current, type.toStringUpper() + " " + name
                             + " was originally defined at " + old + ".");
        else
            return new Error(current, type.toStringUpper() + " " + name
                             + " was originally defined (as " + oldType.toStringArticle() + ") at" + old + ".");
    }
```

```
    /**
     * Generated when a proc is referenced, but never defined.
     *
     * @param name name of proc
     * @param current location of reference
     * @param type context of reference
     * @return an Error object
     */
    public static Error procNotDefined(String name, Loc current, Proc.ProcType type)
    {return new Error(current, type.toStringUpper() + " " + name + " has not been defined.");}

    /**
     * Generated when a state is referenced, but never defined.
     *
     * @param name name of state
     * @param current location of reference
     * @return an Error object
     */
    public static Error stateNotDefined(String name, Loc current)
    {return new Error(current, "State " + name + " has not been defined.");}

    /**
     * Generated when a function is called as an action, or vice versa.
     *
     * @param name name of proc
     * @param type context of reference
     * @param current location of reference
     * @param oldType context of original declaration
     * @param old location of declaration
     * @return an Error object
     */
    public static Error procContextInconsistent(String name, Proc.ProcType type, Loc current,
                                                Proc.ProcType oldType, Loc old)
    {return new Error(current, name + " is called as " + type.toStringArticle()
                      + ", but was defined as " + oldType.toStringArticle()
                      + " (at " + old + ".)");}

    /**
     * Generated when a proc is called with the wrong number of arguments.
     *
     * @param name name of proc
     * @param type context of reference
     * @param numArgs number of arguments used in reference
     * @param current location of reference
     * @param oldNumArgs declared number of arguments
     * @param old location of declaration
     * @return an Error object
     */
    public static Error procWrongNumberOfArgs(String name, Proc.ProcType type, int numArgs,
                                              Loc current, int oldNumArgs, Loc old)
    {return new Error(current, name + " is called with " + numArgs
                      + " arguments, but takes " + oldNumArgs
                      + " arguments (as declared at " + old + ".");}

    /**
     * Generated when a proc implementation is not valid.
     *
     * @param current location of declaration
     * @param className name of Java class specified
     * @param type context of declaration
     * @return an Error object
     */
    public static Error badImplementation(Loc current, String className, Proc.ProcType type)
    {return new Error(current, className + " is not a valid " + type + " implementation.");}

    /**
     * Generated when a proc implementation cannot be found.
     *
     * @param current location of declaration
     * @param className name of Java class specified
     * @return an Error object
     */
    public static Error implementationNotFound(Loc current, String className)
    {return new Error(current, "Class " + className + " could not be loaded.");}

}
```

A.2.9 isicl/Loc.java

```
package isicl;

/**
 * Stores a location in a source file.
 * This should be simple, but there's a lot of garbage
 * needed to compensate for the fact that ANTLR's
 * Token.getFilename() apparently doesn't work.
 *
 * @author  Matthew Keitz
 */

import antlr.collections.AST;

class Loc implements Comparable {
    private int line, column, fileOrder;
    private String file;

    private static String currentFile;
    private static int currentFileOrder;

    /**
     * Sets the filename for all Loc's created henceforth.
     *
     * @param name name of file
     */
    public static void setFilename(String name) {
        currentFile = name;
        currentFileOrder++;
    }

    /**
     * Resets the count of files processed.
     */
    public static void reset()
    {currentFileOrder = 0;}

    /**
     * Returns a special, dummy location, for
     * system-generated errors.
     *
     * @return a system location
     */
    public static Loc system() {
        Loc result = new Loc(0,0);
        result.fileOrder = 0;
        result.file = "system";

        return result;
    }

    /**
     * Constructs a new Loc, with the given location.
     * File name is set automatically.
     *
     * @param line line in source file
     * @param column column in source file
     */
    public Loc(int line, int column) {
        this.line = line;
        this.column = column;
        file = currentFile;
        fileOrder = currentFileOrder;
    }

    /**
     * Defines an order between Loc objects.
     *
     * @param x another Loc object
     * @return 1 if this Loc is after the other; -1
     * if it is before; 0 if they are equal
     */
    public int compareTo(Object x) {
        if (fileOrder > ((Loc)x).fileOrder)
            return 1;
        else if (fileOrder < ((Loc)x).fileOrder)
```

```
            return -1;
        else if (line > ((Loc)x).line)
            return 1;
        else if (line < ((Loc)x).line)
            return -1;
        else if (column > ((Loc)x).column)
            return 1;
        else if (column < ((Loc)x).column)
            return -1;

        return 0;
    }

    public String toString()
    {return file + ":" + line + ":" + column;}
}
```

A.2.10 isicl/ExtendedAST.java

```
package isicl;

/**
 * A replacement AST node class. The standard AST node
 * (oddly) does not retain line and column information.
 *
 * @author Matthew Keitz
 */

import antlr.CommonAST;
import antlr.Token;

public class ExtendedAST extends CommonAST {
    private Loc loc;

    /**
     * Stores the location, and calls the regular
     * initialize routine.
     *
     * @param t this node's data
     */
    public void initialize(Token t) {
        super.initialize(t);

        loc = new Loc(t.getLine(), t.getColumn());
    }

    /**
     * Gets the location.
     *
     * @return the location
     */
    public Loc getLoc()
    {return loc;}
}
```

A.3 Runtime

A.3.1 isicl/standard.icl

```
// Predefined Actions.

action ahead: isicl.runtime.actions.Ahead

action back: isicl.runtime.actions.Back

action right: isicl.runtime.actions.Right

action left: isicl.runtime.actions.Left

action stop: isicl.runtime.actions.Stop
```

```
action resume: isicl.runtime.actions.Resume

action fire: isicl.runtime.actions.Fire

action pass: isicl.runtime.actions.Pass

// Predefined flags.

//function hit_enemy: isicl.runtime.functions.HitEnemy

//function hit_bullet: isicl.runtime.functions.HitBullet

//function missed: isicl.runtime.functions.Missed

//function hit_self: isicl.runtime.functions.HitSelf

//function bump_enemy: isicl.runtime.functions.BumpEnemy

//function bump_wall: isicl.runtime.functions.BumpWall

//function scanned_enemy: isicl.runtime.functions.ScannedEnemy

//function robot_died: isicl.runtime.functions.RobotDied

// Predefined information functions.

function xpos: isicl.runtime.functions.Xpos

function ypos: isicl.runtime.functions.Ypos

function heading: isicl.runtime.functions.Heading

function speed: isicl.runtime.functions.Speed

function map_height: isicl.runtime.functions.MapHeight

function map_width: isicl.runtime.functions.MapWidth

function enemies: isicl.runtime.functions.Enemies

function energy: isicl.runtime.functions.Energy

// Predefined sensor functions.

//function enemy_direction: isicl.runtime.functions.EnemyDirection

//function enemy_distance: isicl.runtime.functions.EnemyDistance

//function enemy_energy: isicl.runtime.functions.EnemyEnergy

//function enemy_heading: isicl.runtime.functions.EnemyHeading

// Predefined utility functions.

function true: isicl.runtime.functions.True

function false: isicl.runtime.functions.False

function sin: isicl.runtime.functions.Sin

function cos: isicl.runtime.functions.Cos

function tan: isicl.runtime.functions.Tan

function floor: isicl.runtime.functions.Floor

function pow: isicl.runtime.functions.Pow
```

A.3.2 isicl/runtime/ISICLRobot.java

```
package isicl.runtime;

import java.util.*;
import robocode.*;
```

```
public class ISICLRobot extends Robot {

    protected StateEngine s;
    protected Vector tv;

    //public ISICLRobot() {}

    public void onBulletHit(BulletHitEvent event) {
        s.clearFlags();
        s.bulletHit = true;
    }
    public boolean getBulletHit()
    {return s.bulletHit;}

    public void onBulletHitBullet(BulletHitBulletEvent event) {
        s.clearFlags();
        s.bulletHitBullet = true;
    }
    public boolean getBulletHitBullet()
    {return s.bulletHitBullet;}

    public void onBulletMissed(BulletMissedEvent event) {
        s.clearFlags();
        s.bulletMissed = true;
    }
    public boolean getBulletMissed()
    {return s.bulletMissed;}

    public void onHitByBullet(HitByBulletEvent event) {
        s.clearFlags();
        s.hitByBullet = true;
    }
    public boolean getHitByBullet()
    {return s.hitByBullet;}

    public void onHitRobot(HitRobotEvent event) {
        s.clearFlags();
        s.hitRobot = true;
    }
    public boolean getHitRobot()
    {return s.hitRobot;}

    public void onHitWall(HitWallEvent event) {
        s.clearFlags();
        s.hitWall = true;
    }
    public boolean getHitWall()
    {return s.hitWall;}

    private double enemyDistance = 0;
    private double enemyEnergy = 0;
    private double enemyHeading = 0;
    private double enemyBearing = 0;
    public void onScannedRobot(ScannedRobotEvent event) {
        s.clearFlags();
        s.scannedRobot = true;
        enemyDistance = event.getDistance();
        enemyEnergy = event.getEnergy();
        enemyHeading = event.getHeading();
        enemyBearing = event.getBearing();
    }
    public boolean getScannedRobot()
    {return s.scannedRobot;}
    public double getEnemyDistance()
    {return enemyDistance;}
    public double getEnemyBearing()
    {return enemyBearing;}
    public double getEnemyHeading()
    {return enemyHeading;}
    public double getEnemyEnergy()
    {return enemyEnergy;}

    public void onRobotDied(RobotDeathEvent event) {
        s.clearFlags();
        s.robotDied = true;
    }
    public boolean getRobotDied()
```

```
    {return s.robotDied;}
}
```

A.3.3 isicl/runtime/AbstractState.java

```
package isicl.runtime;

import robocode.*;
import java.util.*;

/**
 * <p>Abstract class representing an ISICL state.</p>
 * <p>Every state specified by the user in an ISICL program
 * translates into a concrete Java subclass of the State class.
 * The code for the state actions translates into the implementation
 * of the <code>actions()</code> method. Similarly, the code for the
 * state transitions translates into a vector of condition-transition
 * pairs.</p>
 *
 * @author  Mark Berman
 * @author  Michael Marcus
 * @version $Id: AbstractState.java,v 1.6 2003/05/13 06:09:43 mlm211 Exp $
 * @see     ConditionPair
 */
public abstract class AbstractState
{
    public AbstractState (String name, Vector transitionVector){}
     /**
     * The actions to be executed within the state.
     */
    public abstract void actions();

    /**
     * Description: Adds a condition-transition pair to the state.
     * @see ConditionPair
     */
    public void addTransition( ConditionPair inCondition )
    {
        transitions.add( inCondition );
    }

    /**
     * Description: Determines the next state based on the current condition
     * and its corresponding transition.
     * @return The name of the next state.
     */
    public String findNextState()
    {
        ConditionPair tempTransition;
        String returnState = stateName;

        for( int i=0; i<transitions.size(); i++ ) {
            tempTransition = (ConditionPair)transitions.get(i);
            if( tempTransition.condition() ) {
                returnState = tempTransition.destinationState;
                i = transitions.size();
            }
        }
        return returnState;
    }

    /**
     * The name of the state.
     */
    public String stateName;

    /**
     * Description: A vector containing condition-transition pairs for the state.
     * @see Vector, ConditionPair
     */
    protected Vector transitions;
}
```

A.3.4 isicl/runtime/StateEngine.java

```
package isicl.runtime;

import robocode.*;
import java.util.*;

/**
 * <p>The ISICL state engine.</p>
 * <p>Keeps track of ISICL states for one robot. During battle,
 * the engine runs through the state machine and feeds actions
 * to the robot based on battle conditions as specified by the
 * user.</p>
 *
 * @author  Mark Berman
 * @author  Michael Marcus
 * @version $Id: StateEngine.java,v 1.3 2003/05/10 04:29:48 mlm211 Exp $
 * @see     AbstractState
 * @see     ConditionPair
 */
public class StateEngine
{
    /**
     * Creates a new default StateEngine object with no initial state.
     */
    public StateEngine()
    {
        stateTable = new Hashtable();
        initialStateName = null;
        currentState = null;
    }

    /**
     * Creates a new StateEngine object with the specified initial state.
     * @param initState The name of the initial state.
     */
    public StateEngine( String initState )
    {
        stateTable = new Hashtable();
        initialStateName = initState;
        currentState = null;
    }

    /**
     * Sets the initial state to the state with the specified name.
     * @param newState The name of the new initial state.
     */
    public void setInitState( String newState )
    {
        initialStateName = newState;
        currentState = null;
    }

    /**
     * Adds a state to the state table.
     * @param newState The new state object.
     * @see State
     */
    public void addState( AbstractState newState )
    {
        stateTable.put( newState.stateName, newState );
    }

    /**
     * Transitions from the current state to the next state as specified
     * in the state transitions of the current state.
     * If the current state is <code>null</code>, the next state is the
     * initial state.
     */
    public void tick()
    {
        if( currentState == null ) {
            currentState = (AbstractState)stateTable.get( initialStateName );
        }
        else {
            currentState = (AbstractState)stateTable.get( currentState.findNextState() );
        }
```

```
            currentState.actions();
        }

        /**
         * Clears all boolean flags.
         */
        public void clearFlags()
        {
            bulletHit = false;
            bulletHitBullet = false;
            bulletMissed = false;
            hitByBullet = false;
            hitRobot = false;
            hitWall = false;
            scannedRobot = false;
            robotDied = false;
        }

        /**
         * The state table. State objects are indexed by their name.
         * @see Hashtable
         */
        protected Hashtable stateTable;

        /**
         * The current state.
         * @see AbstractState
         */
        protected AbstractState currentState;

        /**
         * The name of the initial (start) state.
         */
        protected String initialStateName;

        /**
         * Flag that is <code>true</code> when the robot's bullet hits an enemy.
         */
        public boolean bulletHit;

        /**
         * Flag that is <code>true</code> when the robot's bullet hits another bullet.
         */
        public boolean bulletHitBullet;

        /**
         * Flag that is <code>true</code> when the robot's bullet hits a wall.
         */
        public boolean bulletMissed;

        /**
         * Flag that is <code>true</code> when the robot is hit by a bullet.
         */
        public boolean hitByBullet;

        /**
         * Flag that is <code>true</code> when the robot hits another robot.
         */
        public boolean hitRobot;

        /**
         * Flag that is <code>true</code> when the robot hits a wall.
         */
        public boolean hitWall;

        /**
         * Flag that is <code>true</code> when the robot detects an enemy nearby.
         */
        public boolean scannedRobot;

        /**
         * Flag that is <code>true</code> when the robot dies.
         */
        public boolean robotDied;
    }
```

A.3.5 isicl/runtime/ConditionPair.java

```
package isicl.runtime;

import robocode.*;

/**
 * Specifies a condition-transition pair for a state.
 *
 * @author  Mark Berman
 * @author  Michael Marcus
 * @version $Id: ConditionPair.java,v 1.4 2003/05/10 04:29:48 mlm211 Exp $
 */
public abstract class ConditionPair
{
    /**
     * Constructor.
     */
    public ConditionPair( String name ) {}

    /**
     * Description: A condition. If <code>true</code> is returned, the robot should
     * transition to the destination state.
     * @return <code>true</code> if the condition is met.
     * @see #destinationState
     */
    public abstract boolean condition();

    /**
     * The name of the destination state.
     */
    protected String destinationState;
}
```

A.3.6 isicl/runtime/actions/Ahead.java

```
package isicl.runtime.actions;

/**
 * Implements ahead.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Ahead implements isicl.interfaces.ActionImplementation {
    public void callAction(robocode.Robot bot, double[] args)
    {bot.ahead(args[0]);}

    public int getNumArgs() {return 1;}
}
```

A.3.7 isicl/runtime/actions/Back.java

```
package isicl.runtime.actions;

/**
 * Implements back.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Back implements isicl.interfaces.ActionImplementation {
    public void callAction(robocode.Robot bot, double[] args)
    {bot.back(args[0]);}

    public int getNumArgs() {return 1;}
}
```

A.3.8 isicl/runtime/actions/Fire.java

```
package isicl.runtime.actions;
```

```
/**
 * Implements fire.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Fire implements isicl.interfaces.ActionImplementation {
    public void callAction(robocode.Robot bot, double[] args)
    {bot.fire(args[0]);}

    public int getNumArgs() {return 1;}
}
```

A.3.9 isicl/runtime/actions/Left.java

```
package isicl.runtime.actions;

/**
 * Implements left.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Left implements isicl.interfaces.ActionImplementation {
    public void callAction(robocode.Robot bot, double[] args)
    {bot.turnLeft(args[0]);}

    public int getNumArgs() {return 1;}
}
```

A.3.10 isicl/runtime/actions/Pass.java

```
package isicl.runtime.actions;

/**
 * Implements pass.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Pass implements isicl.interfaces.ActionImplementation {
    public void callAction(robocode.Robot bot, double[] args)
    {}

    public int getNumArgs() {return 0;}
}
```

A.3.11 isicl/runtime/actions/Resume.java

```
package isicl.runtime.actions;

/**
 * Implements resume.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Resume implements isicl.interfaces.ActionImplementation {
    public void callAction(robocode.Robot bot, double[] args)
    {bot.resume();}

    public int getNumArgs() {return 0;}
}
```

A.3.12 isicl/runtime/actions/Right.java

```
package isicl.runtime.actions;

/**
```

```
 * Implements right.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Right implements isicl.interfaces.ActionImplementation {
    public void callAction(robocode.Robot bot, double[] args)
    {bot.turnRight(args[0]);}

    public int getNumArgs() {return 1;}
}
```

A.3.13 isicl/runtime/actions/Stop.java

```
package isicl.runtime.actions;

/**
 * Implements stop.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Stop implements isicl.interfaces.ActionImplementation {
    public void callAction(robocode.Robot bot, double[] args)
    {bot.stop();}

    public int getNumArgs() {return 0;}
}
```

A.3.14 isicl/runtime/functions/BumpEnemy.java

```
package isicl.runtime.functions;

/**
 * Implements bump_enemy.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */

import isicl.runtime.ISICLRobot;

public class BumpEnemy implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args) {
        if (((ISICLRobot)bot).getHitRobot())
            return 1;
        else
            return 0;
    }

    public int getNumArgs() {return 0;}
}
```

A.3.15 isicl/runtime/functions/BumpWall.java

```
package isicl.runtime.functions;

/**
 * Implements bump_wall.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */

import isicl.runtime.ISICLRobot;

public class BumpWall implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args) {
        if (((ISICLRobot)bot).getHitWall())
            return 1;
        else
```

```
            return 0;
    }

    public int getNumArgs() {return 0;}
}
```

A.3.16 isicl/runtime/functions/Cos.java

```
package isicl.runtime.functions;

/**
 * Implements cos.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Cos implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args)
    {return Math.cos(args[0]);}

    public int getNumArgs() {return 1;}
}
```

A.3.17 isicl/runtime/functions/Enemies.java

```
package isicl.runtime.functions;

/**
 * Implements enemies.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Enemies implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args)
    {return bot.getOthers();}

    public int getNumArgs() {return 0;}
}
```

A.3.18 isicl/runtime/functions/EnemyDirection.java

```
package isicl.runtime.functions;

/**
 * Implements enemy_direction.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */

import isicl.runtime.ISICLRobot;

public class EnemyDirection implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args)
    {return ((ISICLRobot)bot).getEnemyBearing();}

    public int getNumArgs() {return 0;}
}
```

A.3.19 isicl/runtime/functions/EnemyDistance.java

```
package isicl.runtime.functions;

/**
 * Implements enemy_distance.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
```

```
 */

import isicl.runtime.ISICLRobot;

public class EnemyDistance implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args)
    {return ((ISICLRobot)bot).getEnemyDistance();}

    public int getNumArgs() {return 0;}
}
```

A.3.20 isicl/runtime/functions/EnemyEnergy.java

```
package isicl.runtime.functions;

/**
 * Implements enemy_energy.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */

import isicl.runtime.ISICLRobot;

public class EnemyEnergy implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args)
    {return ((ISICLRobot)bot).getEnemyEnergy();}

    public int getNumArgs() {return 0;}
}
```

A.3.21 isicl/runtime/functions/EnemyHeading.java

```
package isicl.runtime.functions;

/**
 * Implements enemy_heading.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */

import isicl.runtime.ISICLRobot;

public class EnemyHeading implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args)
    {return ((ISICLRobot)bot).getEnemyHeading();}

    public int getNumArgs() {return 0;}
}
```

A.3.22 isicl/runtime/functions/Energy.java

```
package isicl.runtime.functions;

/**
 * Implements energy.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Energy implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args)
    {return bot.getEnergy();}

    public int getNumArgs() {return 0;}
}
```

A.3.23 isicl/runtime/functions/Floor.java

```
package isicl.runtime.functions;

/**
 * Implements floor.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Floor implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args)
    {return Math.floor(args[0]);}

    public int getNumArgs() {return 1;}
}
```

A.3.24 isicl/runtime/functions/Heading.java

```
package isicl.runtime.functions;

/**
 * Implements heading.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Heading implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args)
    {return bot.getHeading();}

    public int getNumArgs() {return 0;}
}
```

A.3.25 isicl/runtime/functions/HitBullet.java

```
package isicl.runtime.functions;

/**
 * Implements hit_bullet.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */

import isicl.runtime.ISICLRobot;

public class HitBullet implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args) {
        if (((ISICLRobot)bot).getBulletHitBullet())
            return 1;
        else
            return 0;
    }

    public int getNumArgs() {return 0;}
}
```

A.3.26 isicl/runtime/functions/HitEnemy.java

```
package isicl.runtime.functions;

/**
 * Implements hit_enemy.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */

import isicl.runtime.ISICLRobot;

public class HitEnemy implements isicl.interfaces.FunctionImplementation {
```

```
    public double callFunction(final robocode.Robot bot, double[] args) {
        if (((ISICLRobot)bot).getBulletHit())
            return 1;
        else
            return 0;
    }

    public int getNumArgs() {return 0;}
}
```

A.3.27 isicl/runtime/functions/HitSelf.java

```
package isicl.runtime.functions;

/**
 * Implements hit_self.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */

import isicl.runtime.ISICLRobot;

public class HitSelf implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args) {
        if (((ISICLRobot)bot).getHitByBullet())
            return 1;
        else
            return 0;
    }

    public int getNumArgs() {return 0;}
}
```

A.3.28 isicl/runtime/functions/MapHeight.java

```
package isicl.runtime.functions;

/**
 * Implements map_height.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class MapHeight implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args)
    {return bot.getBattleFieldHeight();}

    public int getNumArgs() {return 0;}
}
```

A.3.29 isicl/runtime/functions/MapWidth.java

```
package isicl.runtime.functions;

/**
 * Implements map_width.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class MapWidth implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args)
    {return bot.getBattleFieldWidth();}

    public int getNumArgs() {return 0;}
}
```

A.3.30 isicl/runtime/functions/Missed.java

```
package isicl.runtime.functions;

/**
 * Implements missed.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */

import isicl.runtime.ISICLRobot;

public class Missed implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args) {
        if (((ISICLRobot)bot).getBulletMissed())
            return 1;
        else
            return 0;
    }

    public int getNumArgs() {return 0;}
}
```

A.3.31 isicl/runtime/functions/Pow.java

```
package isicl.runtime.functions;

/**
 * Implements pow.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Pow implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args)
    {return Math.pow(args[0], args[1]);}

    public int getNumArgs() {return 2;}
}
```

A.3.32 isicl/runtime/functions/Random.java

```
package isicl.runtime.functions;

/**
 * Implements random.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Random implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args) {
        if (Math.random() < args[0])
            return 1;
        else
            return 0;
    }

    public int getNumArgs() {return 1;}
}
```

A.3.33 isicl/runtime/functions/RobotDied.java

A.3.34 isicl/runtime/functions/ScannedEnemy.java

A.3.35 isicl/runtime/functions/Sin.java

```
package isicl.runtime.functions;

/**
 * Implements sin.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Sin implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args)
    {return Math.sin(args[0]);}

    public int getNumArgs() {return 1;}
}
```

A.3.36 isicl/runtime/functions/Speed.java

```
package isicl.runtime.functions;

/**
 * Implements speed.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Speed implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args)
    {return bot.getVelocity();}

    public int getNumArgs() {return 0;}
}
```

A.3.37 isicl/runtime/functions/Tan.java

```
package isicl.runtime.functions;

/**
 * Implements tan.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Tan implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args)
    {return Math.tan(args[0]);}

    public int getNumArgs() {return 1;}
}
```

A.3.38 isicl/runtime/functions/Xpos.java

```
package isicl.runtime.functions;

/**
 * Implements xpos.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Xpos implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args)
    {return bot.getX();}

    public int getNumArgs() {return 0;}
}
```

A.3.39 isicl/runtime/functions/Ypos.java

```
package isicl.runtime.functions;

/**
 * Implements ypos.
 * See language reference manual for documentation.
 *
 * @author Matthew Keitz
 */
public class Ypos implements isicl.interfaces.FunctionImplementation {
    public double callFunction(final robocode.Robot bot, double[] args)
    {return bot.getY();}

    public int getNumArgs() {return 0;}
}
```

A.4 Interfaces

A.4.1 isicl/interfaces/ActionImplementation.java

```
package isicl.interfaces;

/**
 * The ActionImplementation interface.
 * All user-defined actions must be contained within a Java class
 * that implements this interface.
 *
 * @author Matthew Keitz
 * @author Michael Marcus
 * @version $Id: ActionImplementation.java,v 1.3 2003/05/12 21:57:34 msk2102 Exp $
 */
public interface ActionImplementation {

    /**
     * Implementation of the action in Java.
     * The user does not call this method directly.  Rather, the method call
     * is generated at compile time by the ISICL compiler.  It is the programmer's
     * responsibility to implement <code>callAction</code> properly when writing
     * actions.
     * @param bot  the robot taking the action
     * @param args the arguments to the action (may be empty)
     * @see   <a href="http://robocode.alphaworks.ibm.com/docs/robocode/" target="_top">robocode</a>
     */
    public abstract void callAction(robocode.Robot bot, double args[]) throws Exception;

    /**
     * Called by the ISICL compiler (at compile time to verify the number
     * of arguments that the action takes.
     *
     * @return the number of arguments
     */
    public abstract int getNumArgs();
}
```

A.4.2 isicl/interfaces/FunctionImplementation.java

```
package isicl.interfaces;

/**
 * The FunctionImplementation interface.
 * All user-defined functions must be contained within a Java class
 * that implements this interface.
 *
 * @author Matthew Keitz
 * @author Michael Marcus
 * @version $Id: FunctionImplementation.java,v 1.3 2003/05/12 21:57:34 msk2102 Exp $
 */
public interface FunctionImplementation {

    /**
     * Implementation of the function in Java.
```

```
     * The user does not call this method directly. Rather, the method call
     * is generated at compile time by the ISICL compiler. It is the programmer's
     * responsibility to implement <code>callFunction</code> properly when writing
     * functions.
     * @param  bot  the robot that is running
     * @param  args the arguments to the action (may be empty)
     * @return the evaluated result of the code inside the method body
     * @see    <a href="http://robocode.alphaworks.ibm.com/docs/robocode/" target="_top">robocode</a>
     */
    public abstract double callFunction(final robocode.Robot bot, double args[]) throws Exception;

    /**
     * Called by the ISICL compiler (at compile time to verify the number
     * of arguments that the function takes.
     *
     * @return the number of arguments
     */
    public abstract int getNumArgs();
}
```