[W]izard of [HO]use [M]anagement

Programming Language and Translators Project
December 2003

Rui Kuang
Arvid Bessen
Andrey Butov
Svetlana Starshinina

Table of Contents:

1.0 Introduction

The WHOM language is designed to allow developers to simulate the creation and management of a household. Using WHOM, it is possible to define a household as well as a set of rules to govern the automatic operations of that household.
WHOM is designed to be a simple but powerful, object-oriented, event-driven, flexible, and robust language.

1.1 Overview

Many sci-fi writers envision a future world in which most of the household tasks would be performed by machines. Even today many objects exist that have the capacity of being programmed. Examples include VCRs, alarm clocks, etc**.** It is not so hard to imagine that in the future many household objects, from windows to full-featured security systems will come with built-in computers capable of controlling various built-in functionalities. A simple window, equipped with the right mechanism, can be easily opened or closed, and locked or unlocked automatically without requiring any action by a human. The possibilities are virtually endless, and could involve not only windows that open and close, but doors that lock and unlock, and refrigerators that automatically restock themselves by contacting the nearest supermarket and ordering the necessary foods to be delivered at a certain time. With a slight stretch of imagination, it is also possible to imagine that in the future objects would be able to perform more complicated tasks, such as windows washing themselves, beds making themselves, and entire households being able to implement internal security and ambiance features. We have assumed that our objects are “futuristic” with complex mechanisms that enable them to perform a variety of function. All objects in the house are connected to a single Household Management System (HMS), which allows them to communicate.

The WHOM language allows to define the behavior of these objects. Instead of issuing commands to each of these objects, the WHOM language is intended to provide a uniform and easy-to-use programming interface to the objects in the house. This way the objects will be able to act even when the owner of the house is not present at all. Thus, rather than going into the trouble of issuing direct commands, the user would be able to write a program in the WHOM language to specify what the objects should do, and in response to what events.

Considering the fact that a majority of the population has no programming experience, we had to make the language very simple to use. In order to use the language, first the house itself needs to be defined and then all the objects inside the house need to be “plugged in”. This may be done by a WHOM system programmer, similar to how some things get installed by the electrician currently. However, the language is simple enough for the user, to be able to define behavior by first issuing simple commands (such as “close window when temperature < 70”), which can be combined to compose a more intelligent household management system.

1.2 Key Characteristics

Simple:

WHOM is very easy to use for users, even those that have minimal programming experience. The commands are very close to the English language and the syntax is very intuitive. The language provides several built-in classes, which can be very easily manipulated by the user by issuing commands such as “close”, “lock”, etc. The object status is periodically examined and appropriate functions can be performed to change the status of the object. Behavior intrinsic to the various objects allows the user to be able to perform simple household tasks with great ease. Additionally the language supports the measurement of several environmental variables, such as temperature, light intensity, etc, and allows the user to specify the appropriate preset or customized response to changes in the environment.

```
// define house
Window northWindow;
Window southWindow;

northWindow.open();
southWindow.close();
```

Powerful:

WHOM is very powerful even for beginners. For more advanced users WHOM presents virtually no limits. In addition to the built-in classes, the users can specify their own classes, with their own set of parameters and commands. Adding a new object type to the household management system will be equivalent to including a library class in a Java program, and then all the commands to manipulate the object will automatically become available to the user. The users can also create new functions in order to simplify more frequent tasks. It is possible to program the HMS to respond to such complicated events like when the kids come home from school, or to program the system so that the house cares for itself while the family leaves for vacation.

```
// define house
Window northWindow;
Window southWindow;
Alarm alarm;

once (northWindow.broken )
{
      alarm.trigger();
}
```

Object-oriented:

The language is heavily object-oriented, since most of the commands in WHOM are commands to objects to perform certain actions. In order for an object to execute a command, it has to be defined for that object. If a certain command does not exist, an error message will be issued. The various objects in a system will be “plugged-in” into the house management system. The objects will be either standard and predefined or customized and defined by the user, similar to existing libraries and self-defined classes in Java. The standard objects, such as windows, doors, TVs will have a predefined set of actions that could be performed on them. The customized objects may come with a more complex set of actions that need to be specified by the user.

Extendible:

The user can specify any kind of house, with any number of rooms, windows, doors and other objects. Any object can be added or removed from the home management system at any time. The actions performed by objects may be scheduled or event driven as desired by the user. It is possible to expand the functionality greatly by introducing third-party libraries such as a home-bot for performing more complex actions with other objects. This home-bot would only need to be specified as an object that is able to perform a great variety of actions using various objects, with a specified movement algorithm. It is also possible to create various preset modes (to help the user using the new household management system) for buildings such as prisons, family homes or kindergartens.

```
class AdjustableLamp extends Light
{ //inherits all methods and variables from Light
          void selfAdjust()
      {  //additional method defined in AdjustableLamp
          light_level = (LIGHTNESS/10.0)*3.0;
      }
}
```

Event-driven:

The entire household management system will be event-driven, with a capability of responding to complex events such as a person entering the room, a robber trying to break into the house, or kids coming home from school. We assumed that some of the objects would have complex movement-detecting, face scanning mechanisms to be able to respond to these kinds of events. For example, the home management system will never limit the actions of the adults in the house, but it may limit the actions of the children (not let them watch TV if they did not do their homework). The events will serve as triggers for the objects to perform a specific kind of action, which could involve performing a direct action, or sending some specific information both to the user or to an emergency telephone number (such as 911)

Robust:

WHOM allows for writing programs that are simple, with extensive error checking being done both at compile-time (by our WHOM compiler and javac) and at run-time. If the user attempts to issue a command to an object that does not exist, he/she will be informed at compile time. If the user attempts to issue an invalid command to an object he/she will be informed at runtime by a message from our simulation system.

2.0 Language Tutorial

2.1 Overview

WHOM programs consist of three parts: (1) class definition, (2) house definition and (3) event handling. In the *class definition* section of the program the user can create entirely new classes or create classes that extend the library classes. In the *house definition* section of the program the user can define house objects using the classes from the library files and classes defined in the class definition section. Finally, in the *event handling* section of the program the user can specify how the objects will react to the realtime variables and to the events that are associated with specific objects.

2.2 Class Definition

Class definition section is optional, and needs to be written only if the user wants to create new classes. Most of the objects, along with their attributes, methods and events, are specified in a standard library file whom.wl. The standard library contains objects that include Room, Window, Blinds, Door, Bed, Table, Chair, Light, Bathtub, Faucet, Oven, Microwave, AirConditioner, Computer, LaundryMachine, AlarmClock, Refrigerator, Fan, Television, VCR, Telephone, FireAlarm, Lawn, Pool, Gate.

For each new class, the user needs to specify its attributes, methods and events. Let's suppose we want to write a class to specify a special kind of lamp: one that adjusts its brightness in response to surrounding level of light. This lamp inherits all the attributes, methods and events from a "standard" lamp, which is already specified in whom.wl, and specifies its own attributes, methods and events.

To declare a class, use the following syntax:

`class` *class_name* [`extends` *class_name*]

Class declaration:

```
class AdjustableLamp extends Light
```

Then we need to specify any attributes, methods and events that are not already specified in the parent class Light. The parent class Light only specifies whether the light is lit or unlit, and whether the light works, but does not specify the brightness of the bulb, so we need an attribute that would store the brightness. We also need to specify each attributes default value, which is initialized when an instance of the class is created.

To specify an attribute use the following syntax:

attribute_type attribute_name = default_value

Attribute specification:

```
number brightness = 0;
```

After we specified the attributes, we need to specify all additional methods. This is how we would write a method that would adjust the brightness of the light in response to the light level in the room.

Use the following syntax to specify a method:

return_type method_name ([attributes])

Method specification:

```
void SelfAdjust()

{

      brightness = (LIGHTNESS/10.0)*3.0;

}
```

When this method is called, the realtime variable LIGHTNESS is used to calculate the new brightness level of the light bulb.

In this particular case, we do not need to specify any additional events, but if we did need to create an event, we would simply need to say event_name;

For example, here is an event that is already specified in the parent class Light.

```
EVENT_BULB_BAD;
```

This event is detected by the Light object itself, and the appropriate action is taken.

Complete code for defining an adjustable lamp:

```
class AdjustableLamp extends Light

{

   number brightness = 0;

   void selfAdjust()

   {

      brightness = (LIGHTNESS/10.0)*3.0;
```

```
    }
}
```

2.3 House Definition

After all the classes are specified, the next step is to create a house. To create a house, we need to specify all the objects in the house. Since most of the objects are specified in the standard library, we also need to import the standard library

To import the standard library, use the syntax: import library_name;

In this case, we only need to import the standard library:

```
import whom.wl;
```

Then we need to create all the rooms and the objects that will go in our rooms. To define an instance of a class, use the following syntax:

class_name class_instance_name;

For example, suppose we want to create a dining room with a self-adjustable lamp:

```
Room diningRoom;

Table dinTable;

Chair dinCh1,dinCh2,dinCh3,dinCh4;

Window dinWin1,dinWin2;

AdjustableLamp dinLight;

Door doorDinKit;
```

2.4 Event Handling

Now we finally get to the real WHOM program, one that allows the user to actually manipulate all the objects that were previously defined, and to handle events that are detected. This is the section that will be used most often, unlike the two previous sections, which are only used when the house is defined and new objects are created. In this section we can react to either realtime variables or to events associated with the objects.

If we want our lamp to self-adjust every time the lightness changes, we would have to store some initial value of lightness and then compare the new level of lightness to the old. In the event handling section we can define a new variable and set it equal to lightness:

```
number oldLightness = LIGHTNESS;
```

Then, we can write a statement that will compare the old lightness level to the new lightness level and if they are not equal to each other, call a method to adjust the lamp.

To write a once condition, use the following syntax:

`once` (*condition*) { *statement*}

Example once statement:

```
once (oldLightness != LIGHTNESS)

{

      lamp.selfAdjust();

      oldLightness = LIGHTNESS;

}
```

The WHOM program will continuously keep on checking whether the condition is true, and execute the statement as soon as it is found to be true

2.5 Complete Example

```
import whom.wl;

//class specification

class AdjustableLamp extends Light

{

   number brightness = 0;

   void selfAdjust()

   {

      brightness = (LIGHTNESS/10.0)*3.0;

   }
```

```
}


//house definition

Room diningRoom;

Table dinTable;

Chair dinCh;

Window dinWin;

AdjustableLamp dinLight;

Door doorDinKit;


//event handling

number oldLightness = LIGHTNESS;

once (oldLightness != LIGHTNESS)

{

       lamp.selfAdjust();

       oldLightness = LIGHTNESS;

}
```

2.6 Compiling and Running WHOM Programs

To run the program “foo.whom” you execute “start foo.whom”, which in turn calls “java wem.WEM foo.whom”. This invokes the WEM GUI, which opens up the file, and executes all the statements in the file passed.

3.0 Language Reference Manual

3.1 Introduction

The WHOM language is designed to allow developers to simulate the creation and management of a household. Using WHOM, it is possible to define a household as well as a set of rules to govern the automatic operations of that household. WHOM is designed to be a simple but powerful, object-oriented, event-driven, flexible, and robust language.

3.2 Lexical Conventions

There are six kinds of tokens: identifiers, keywords, constants, strings, expression operators, and other separators. In general blanks, tabs, newlines, and comments as described below are ignored except as they serve to separate tokens. At least one of these characters is required to separate otherwise adjacent identifiers and constants

If the input stream has been parsed into tokens up to a given character, the next token is taken to include the longest string of characters, which could possibly constitute a token.

3.2.1 Comments

The characters /* start a comment terminated by the first */, while the characters // start a single-line comment that runs until the end of the line.

3.2.2 Identifiers

An identifier is a sequence of letters and digits; the first character must be alphabetic. The underscore ‘‘_’’ counts as alphabetic. Upper and lower case letters are considered different.

3.2.3 Keywords

The following identifiers are reserved for use as keywords, and may not be used otherwise:

```
number      string
while       return
once        class
void        extends
if          import
else        dispatch
once        realtime
for
```

3.2.4 Constants

There are two types of constants in WHOM: numbers and strings

3.2.4.1 Number

A number consists of an integer part, a decimal point and a fraction part. The integer and fraction parts both consist of a sequence of digits. Either the integer part or the decimal point and the fraction part (not both) may be missing.

3.2.4.2 String

A string is a sequence of characters surrounded by double quotes ‘‘ " ’’. A double quote inside a string is represented by two consecutive double quotes

3.2.5 Other tokens

{ } ( ) , ;

+ - * / =

>= <= == != > <

| ! & .

3.3. Expressions

The WHOM expressions follow the standard precedence and associativity rules

3.3.1 Primary expressions

Primary expressions include identifiers, constants and parenthesized expressions.

3.3.1.1 Identifiers

An identifier itself could be a left-value expression or a right-value expression. It if it is a right value expression it is evaluated.

3.3.1.2. Constants

A constant, whether it is a number or a string, is a right-value expression. It can be evaluated either directly to a constant itself, or using function calls which take the following form:

function_name (argument_list$_{opt}$)

The optional list of arguments contains zero or more constants or identifiers separated by a comma.

3.3.1.3 Parenthesized expressions

Parenthesized expression is a primary expression. Parentheses take precedence over all other operators.

3.3.2 Arithmetic expressions

Arithmetic expressions take primary expressions as operands and evaluate them using operators

3.3.2.1. Unary operators

Unary operators + and – can be prefixed to an expression. The + operator returns the expression itself whereas the – operator returns the negative of the primary expression. These operators are applicable to the constants of type number

3.3.2.2 Binary operators

Binary operators +, -, *, / indicate addition, subtraction, multiplication, division, respectively. Multiplication and division take precedence over addition and subtraction. These operators are applicable to the constants of type number

3.3.3 Relational expressions

Binary relational operators >=, <=, ==, !=, > and < indicate whether the first operand is

greater than or equal to, less than or equal to, equal to, not equal to, greater than, or less than the second operand, respectively. The arithmetic operators take precedence over the relational operators

3.3.4 Logical expressions

Logical operators take relational expressions as operands, so logical expressions have the lowest precedence. Logical operators !, |, & indicate “negate”, “or” and “and”, respectively. Among these operators, operator ! has the highest precedence, then operator &, and operator | has the lowest precedence.

3.4. Statements

Except as indicated, statements are executed in sequence.

3.4.1 Expression statement

Most statements are expression statements, which have the form

expression ;

Usually expression statements are assignments or function calls

3.4.1 Declaration statement

Some statements are used to declare a new variable, which has the form

`[realtime]` *variable_type varible_name;*

where variable_type could be either `number` or `string.`

3.4.2 Compound statements

A group of zero or more statements can be surrounded by { and }, in which case they are treated as a single statement.

3.4.3 Assignments

An assignment assigns a constant to a specified identifier. Assignments take the form:

left_value_expression = right_value_expression;

3.4.4 Conditional statement

The two forms of the conditional statement are

`if (`*expression*`)` *statement*

`if (`*expression*`)` *statement* `else` *statement*

In both cases the expression is evaluated and if it is true, the first substatement is executed. In the second case the second substatement is executed if the expression is false. The dangling ‘‘else’’ ambiguity is resolved by connecting an `else` with the last encountered elseless `if`.

3.4.5 Iterative statements

There are two types of iterative statements, while statements and for statements

3.4.5.1 while statements

The `while` statement has the form

`while (` *expression* `)` *statement*

The substatement is executed repeatedly so long as the value of the expression remains nonzero. The test takes place before each execution of the statement.

3.4.5.2 for statements

The `for` statement has the form

`for (` *expression1*$_{opt}$ `;` *expression2*$_{opt}$ `;` *expression3*$_{opt}$ `)` *statement*

Thus the first expression specifies initialization for the loop; the second specifies a test, made before each iteration, such that the loop is exited when the expression becomes false; the third expression typically specifies an incrementation, which is performed after each iteration.

3.4.6 Method invocation and return

3.4.6.1. Method invocation

Operator . is used to invoke a method of a specified class. Method invocation takes the form:

class_instance_name.method_name([arguments])

where class_instance_name is the name of the instance of the class of which the method is invoked, method_name is the name of the method within that class and arguments is an optional list of values, , separated by commas, matching the parameters of the defined method.

3.4.6.2 Return statement

A function returns to its caller by means of the `return` statement, which has one of the forms:

`return;`

`return` *expression* `;`

In the first case no value is returned. In the second case, the value of the expression is returned to the caller of the function.

3.5. Classes and Realtime Variables

3.5.1 Classes

All objects in the house are represented as instances of classes.

3.5.1.1 Class definition

The user can define classes that extend the pre-defined objects or create entirely new classes. Classes are defined at the beginning of the execution of the WHOM program. Each class inherits the attributes from the standard WHOM superclass and follows the standard WHOM interface. It can implement its own constructor and its own methods. All the attributes, constructor and methods are public. Class definition takes the following form:

`class` *class_name* `extends` *superclass_name*

3.5.1.2 Standard library (whom.wl)

The standard library contains standard classes that have already been implemented (see 3.5.1.6 for a list of standard classes). Standard library and other user-defined libraries need to be imported by the user. Since all the attributes and methods are public, the user is free to change any of the attributes and to use any of the methods. All libraries have .wl extension, and the standard library is stored in the whom.wl file. whom.wl contains most of the classes that the users will need to specify a house. It is specifically created to make the house manipulation much easier. If the user imports whom.wl he or she can use the classes defined in whom.wl without ever needing to specify any additional classes. All the realtime variables are also specified in whom.wl, so it must be imported every time if the realtime variables are to be used in a program.

3.5.1.3 User-defined libraries

The user can define new classes and store them in .wl files. These classes can later be imported and used by other programs.

3.5.1.3 Attributes

Each class can have a set of attributes that could be of type number or string, that contain quantitative or descriptive information about a particular instance of a class

any(class_name).attribute_name;

3.5.1.4 Methods

The class methods are defined in the clasdef section and then called in the event section of the program. The method implementation takes the form:

return_type method_name ([parameters]) { implementation}

3.5.1.5 Events

Each class can have a set of events that applies to all instances of that class. The control system will be notified every time this event occurs. The event declaration takes the following form:

event_name;

3.5.1.6 List of built-in classes

Built-in classes include Room, Window, Blinds, Door, Bed, Table, Chair, Light, Bathtub, Faucet, Oven, Microwave, AirConditioner, Computer, LaundryMachine, AlarmClock,

Refrigerator, Fan, Television, VCR, Telephone, FireAlarm, Lawn, Pool, Gate.

3.5.2 Realtime Variables

Realtime variables specify a set of environmental conditions to which the WHOM objects react. Some realtime variables are standard, but the users can specify their own.

3.5.2.1 Declaration

The declaration of a realtime variable (unless it is a part of a standard list of realtime variables) has the following form:

`realtime` *variable_type variable_name;*

3.5.2.2 Built-in realtime variables

Built-in realtime variables include: temperature, humidity, weather, hours, minute, second, air pressure, noise, precipitation and lightness.

3.6. House Definition

After the classes are specified, the house objects are defined

3.6.1 Importing standard libraries

In order to use the standard libraries, the user needs to import the .wl files. whom.wl should always be imported but other user-defined libraries can also be imported. The importing of a library takes the following form:

`import` *"library1_name.wl"[,"library2_name.wl"…] ;*

3.6.2 Object creation

Object creation (or declaration) takes the following form:

class_name id_name ;

The class_name is a name of a class that is contained either in the standard or user-defined libraries and id_name is a name for a particular instance of a class that is specified by the user

3.7. Scoping

WHOM places a variable in one of three scopes at the time of definition.

The highest resolution scope is the program global scope. Variables in the program global

scope may be referenced and modified in any part of the source code.

The second highest resolution scope is class instance scope. Variables defined within a class, such as class data members have scope spanning the entire class. This means that any method within that class can reference/modify all variables of that class (as well as all variables defined in the program global scope). All methods, attributes and events defined within the class, may be accessed from outside the class by using the . (dot) operator on the instance of that class.

The last resolution of scoping is at the statement block level. Variables defined within a given statement block of a class have the scope of that block. Any expression or statement of that block may reference/modify that variable (as well as variables defined in the class instance scope, as well as in the program global scope).

3.8. Event handling

This portion of the program specifies how the objects will react to the realtime variables and events that are associated with specific objects. Responding to an event takes the following form:

`once` *condition {statement}*

Condition can be either specified by the user directly using the realtime variables or can be a built-in event associated with a specific object or type of object. Statement specifies what needs to be done in response to the event. Conditions are evaluated and actions are carried out whenever variables in the condition or the standard events change.

3.8.1 Responding to realtime variables

Responding to realtime variables involves testing the current conditions of variables against some prespecified conditions. The condition takes the form of a logical expression.

3.8.2 Responding to events associated with objects

Responding to an event that is associated with objects involves checking whether this event is true for a specific object. The condition has the following form:

classname.event_name

3.8.3 Event queue

All the events are placed in the queue and executed in the order that they appear in the program. If a variable that is used as a condition in one event is modified in another event, then the condition is reevaluated and if necessary the event is triggered.

4.0 Project Plan

4.1 General Project Processes

Planning and specification took place in a set of initial meetings in which we decided on the division of work and the specification of the WHOM language. We divided the work into front-end, back-end, testing and documentation and assigned one person to be primarily responsible for one task, and a second person to help with that task. This was done in order increase the productivity of each person while simultaneously focusing on the quality of each part. We used the initial meetings to address the possible difficulties, discarding some ideas that we realized would be difficult to implement.

Development and testing were done in the previously described manner: with one person being responsible for a specific task, and another person helping in that task. Development always oriented towards specific code examples that got increasingly more complicated until in the end all features of WHOM were fully supported. The testing framework (discussed in detail below), was developed in parallel to the development of the language itself.

4.2 Programming Style Guide

The team agreed to use a clear programming style so that any person in the team would be able to understand and debug the code written by another person. This allowed us to maximize the efficiency of each person who wrote the code. The general rules that we followed are listed below. The same programming style was used for both Java and WHOM code.

4.2.1 Commenting

Each class and each method in the class contains appropriate comments that describes the use of this class or a method. We used javadoc to be able to easily view all the comments, and followed javadoc standards for comments

4.2.2 Indentation and spacing:

- Indentation of each level is 4 spaces
- The left brace “{" occupies a full line and is at the same column as the first character of the previous line

- The right brace "}" occupies a full line and at the same column of its corresponding left brace
- Insert a space between arguments and their parentheses "(" ")", but no space between function name and "(".
- Insert spaces between operators and operands for outer expressions.
- For statements such as if…then…else, start each keyword on a separate line

4.2.3 Naming Conventions

- All class names begin with an uppercase character, e.g. Lamp
- All instances of classes, variables and method names begin with lowercase characters, e.g. myLamp
- If there are more than one word in a variable name, all words except the first begin with capital letters
- Make variable names short but understandable
- Objects that are placed in a specific room, should start with the room's name, e.g. dinWin1 means the first window in the dining room

4.3 Project Timeline

From the outset of the project we came up with a comprehensive plan for completing the project on time. We had set several milestones with specific deadlines, set up weekly meetings with specific agendas to discuss any difficulties and provide the team members with regular updates. Here is a timeline that we used for the project

Goal	**Due Date**
Complete the white paper	9/23/2003
Determine the division of responsibilities, agree on most aspects of syntax	9/25/2003
Prepare a simple version of the syntax, create the first version of language specification	9/30/2003
Prepare a draft of the Language Reference Manual, basic WHOM grammar	10/07/2003
Finalize WHOM grammar	10/15/2003
Finish lexer and parser for WHOM language	10/21/2003
Program with basic arithmetic works	10/28/2003
Complete the LRM	10/28/2003
Program with more complex arithmetic and simple events works	11/01/2003
Finish the backend portion	11/20/2003

Complete final testing and present	12/12-19/2003

4.4 Roles and Responsibilities of Team Members

The table below illustrates the breakup of the responsibilities. However, despite the clear breakup of responsibilities it was understood that all team members would help each other out, if needed.

	Front-end	Back-end	Testing	Documentation
Rui Kuang (group leader)	1	3	3	-
Arvid Bessen	-	1	-	-
Andrey Butov	-	3	1	3
Lana Starshinina	-	-	3	1

Key: *1 – primary responsibility*

2 – helper

3 – may help depending on availability

4.5 Software Development Environment

CVS was used for version control. The lexer, parser and AST walker were implemented using ANTLR language tool with Java 1.4.2. The back-end Java code was written in Java 1.4.2

4.6 Project Log

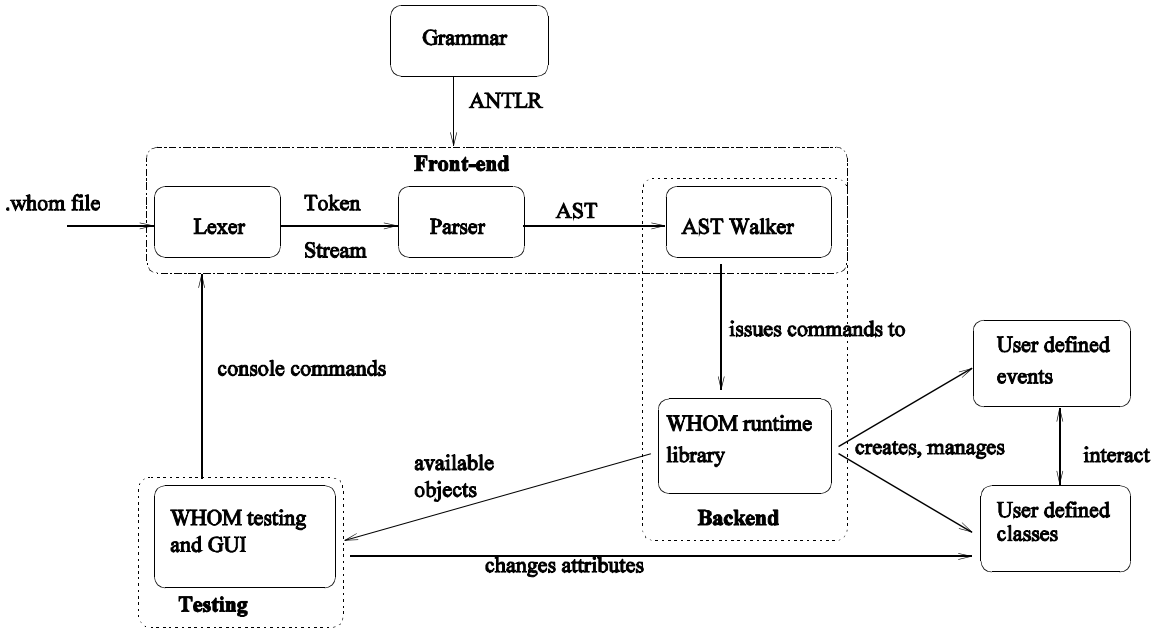
Log Entry	**Date**
Language specification document	9/17/2003
Draft white paper	9/19/2003
Final white paper	9/22/2003
Basic classes implemented	10/06/2003
First version of WEM	10/06/2003
Updated language specification	10/07/2003
Draft of the LRM	10/07/2003
Posted architecture document	10/08/2003
First try at backend implementation	10/10/2003

Code compiles properly	10/12/2003
Some additional code	10/13/2003
Makefiles for easier compilation	10/13/2003
Backend implementation, untested but compiles	10/14/2003
Classes and methods working	10/16/2003
Added support for model/view architecture	10/18/2003
Minor changes and cleanups	10/20/2003
First version of backend complete	10/21/2003
New version of makefiles	10/24/2003
Simple program works	10/27/2003
LRM complete	10/28/2003
Added functionality	11/01/2003
Program with more complex arithmetic and simple events works	11/01/2003
Clean up, removed obsolete files	11/01/2003
WEM starts automatically	11/01/2003
Initial support for WEM-WHOM interface	11/02/2003
Fixed a problem with scooping	11/03/2003
Added some example files, generated docs	11/03/2003
Command line parameter support	11/04/2003
First version of WHOM standard library	11/04/2003
Added support for realtime and observable variables	11/04/2003
Fixed several logic problems	11/06/2003
Scoping works correctly	11/06/2003
Some fixes and additions to the walker	11/07/2003
More changes to the backend, cleanup	11/08/2003
Added proper initialization for global variables	11/09/2003
Full support for real time variables	11/09/2003
Exposed interface to the backend	11/11/2003
Fixed the dangling else problem	11/11/2003
Added constructors, return, string class; fixed bugs	11/13/2003
Fixed the status bar	11/13/2003
First version of error handling	11/13/2003
Added a log screen	11/15/2003
Added reset and clear support	11/15/2003
Restructured code by eliminating model class	11/15/2003
More support for object definitions; added ancestor to attributes	11/15/2003
More simplification and cleanup	11/16/2003
Additional error handling	11/18/2003
Changes from backend visible in GUI	11/19/2003
More resilient object query; display of attributes	11/19/2003
Full requery; objects within objects	11/20/2003
Large cleanup; fixed many bugs	11/20/2003
Support polymorphism	11/21/2003
Non-realtime global variables displayed	11/21/2003
Implemented and tested timers	11/23/2003
Error handling, cleanup	11/24/2003

Recursion support	11/25/2003
Separated scopes into instantiated and parsing	11/27/2003
Static semantic analysis, bug fixes, documentation	12/02/2003
Standard library changes	12/05/2003
Added support for realtime strings	12/07/2003
Example for presentation	12/10/2003
Updated comments and documentation	12/13/2003

5.0 Architectural Design

5.1 Block Diagram of WHOM Interpreter



5.2 Description of Architecture

There are 3 main components of the WHOM interpreter: front-end, backend and the GUI.

5.2.1 Front-end

The source .whom file is first passed to the lexer, which produces a token stream. This token stream is parsed by the parser according to the rules specified in the WHOM grammar. The parser constructs an Abstract Syntax Tree, which is sent to the AST walker at the backend. The parser also checks for syntactical errors and outputs them to the log file in the GUI.

5.2.2 Backend

At the backend AST walker goes through the nodes in the AST and executes the code by issuing commands to the WHOM runtime library. WHOM runtime library creates and manages user-defined classes and user-defined events, which interact with one another. The WHOM runtime library sends the information about available objects to the GUI, where these objects are displayed

Hierarchy For Package whom.backend

Class Hierarchy

- class java.lang.Object
  - class whom.backend.**Attribute**
  - class whom.backend.**Class**
    - class whom.backend.**RootClass**
    - class whom.backend.**Event** (implements java.util.Observer)
    - class whom.backend.**Expression** (implements whom.backend.DependencyTracker)
    - class whom.backend.**ArithmeticOp**
      - class whom.backend.**Addition**
      - class whom.backend.**Division**
      - class whom.backend.**Multiplication**
      - class whom.backend.**Subtraction**
    - class whom.backend.**BooleanToNumber**
    - class whom.backend.**Condition**
    - class whom.backend.**Comparison**
      - class whom.backend.**ComparisonEQ**
      - class whom.backend.**ComparisonGE**
      - class whom.backend.**ComparisonGT**
      - class whom.backend.**ComparisonLE**
      - class whom.backend.**ComparisonLT**
      - class whom.backend.**ComparisonNE**

- class whom.backend.**ConditionAnd**
- class whom.backend.**ConditionNot**
- class whom.backend.**ConditionOnExpression**
- class whom.backend.**ConditionOr**

- class whom.backend.**Lookup**
  - class whom.backend.**BlockLookup**
  - class whom.backend.**ConstBooleanLookup**
  - class whom.backend.**ConstNumberLookup**
  - class whom.backend.**ConstStringLookup**
  - class whom.backend.**GlobalLookup**
  - class whom.backend.**ObjectMemberLookup**
  - class whom.backend.**ThisLookup**
- class whom.backend.**Methodcall**

- class whom.backend.**Method**
- class java.util.Observable
  - class whom.backend.**ObservableObject** (implements java.lang.Cloneable)
    - class whom.backend.**Boolean** (implements whom.backend.DependencyTracker, whom.backend.LogicValue)
      - class whom.backend.**False**
      - class whom.backend.**True**
    - class whom.backend.**Number** (implements whom.backend.Comparable, whom.backend.DependencyTracker, whom.backend.LogicValue)
      - class whom.backend.**ConstNumber**
    - class whom.backend.**Object** (implements whom.backend.DependencyTracker)

        - class whom.backend.**WhomString** (implements whom.backend.Comparable, whom.backend.DependencyTracker, whom.backend.LogicValue)
        - class whom.backend.**ConstString**
- class whom.backend.**Scope**
    - class whom.backend.**InstantiatedScope**
        - class whom.backend.**BlockScopeInstantiated**
        - class whom.backend.**EmptyScopeInstantiated**
        - class whom.backend.**GlobalScopeInstantiated**
        - class whom.backend.**ObjectScope**
    - class whom.backend.**ParsingScope**
        - class whom.backend.**BlockScope**
        - class whom.backend.**ClassScope**
        - class whom.backend.**EmptyScope**
        - class whom.backend.**GlobalScope**
- class whom.backend.**Statement**
    - class whom.backend.**Assignment**
    - class whom.backend.**AssignmentToVoid**
    - class whom.backend.**Block**
    - class whom.backend.**Dispatch**
    - class whom.backend.**For**
    - class whom.backend.**IfThen**
    - class whom.backend.**IfThenElse**
    - class whom.backend.**Return**
    - class whom.backend.**While**
- class java.lang.Throwable (implements java.io.Serializable)
    - class java.lang.Exception

        - class whom.backend.**NoSuchVariableNameException**
        - class whom.backend.**ReturnException**
    - class whom.backend.**Type**

Interface Hierarchy

- interface whom.backend.**Comparable**
- interface whom.backend.**DependencyTracker**
- interface whom.backend.**LogicValue**

5.2.3 WEM

WEM stands for WHOM emulator. It displays each object in a GUI, along with the information about each objects attributes and events. It allows the user to manually change the attributes and trigger events, and then displays the appropriate response. Attributes can either be entered directly, or as console commands in WHOM, in which case they are sent to the lexer to be executed. When the attributes are changed directly, the change is sent directly to the backend.

6.0 Testing WHOM Functionality

6.1 - Overview

The design of WHOM reflects a certain assumption that the execution environment in which WHOM programs will be executed will be an embedded system. More specifically, the language is designed with the prospect that the system executing a given WHOM program will be part of a living environment, in which the given system is entrusted, to not only manage itself and its own needs, but also to satisfy the various needs of the inhabitants of that environment.

While the original design specifying the procedures for testing the various WHOM systems was a sort of automated 'conveyor belt' approach to facilitate eventual regression testing, it was soon realized that such an approach was not feasible for the sort of testing we wanted done, and the sort of information we wanted to extract from failed test cases.

The testing system for WHOM, was thus developed dynamically, and in parallel, to the development of the WHOM lexical analyzer, parser, abstract syntax tree walker, and the WHOM backend. The following outlines the various stages of WHOM testing, as they were developed and used throughout the design and development of that WHOM language itself.

6.2 - Testing Stages

6.2.1 - WEM

From the beginning, it was evident to us that we needed a tool for testing/debugging WHOM, as well as a tool for displaying the final demos of WHOM program execution. Since it was obviously impractical (assuming it’s even technologically feasible at the moment) to build a full-fledged embedded systems environment which could accommodate all of the features of WHOM which we needed to test, we decided to build a software 'emulator' for both of these needs. The emulator would serve all of our testing needs, and would, as well, be used to eventually demo the results. Moreover if the emulator would proved to be of any value whatsoever its core functionality would need to be implemented and working well before the final stages of WHOM testing began. Thus, the creation of WEM ([W]HOM [EM]ulator) began in parallel to the development of WHOM.

6.2.2 – Temporary Testing Solutions

Initially, after the delegation of responsibilities to each member of the WHOM team, the individual parts of WHOM which were being built in parallel did not yet have any co-

dependencies. This provided us with the time required for WEM to be designed and built. Unfortunately, it also meant that the individual developer would have to 'hack' together his/her own mini tests to debug whatever unit they were working on at the moment. As it turned out, this worked quite well as the individual parts of WHOM could be tested without the 'pollution' of the other components. For example, what would eventually become the main module of the backend had a self sustaining executable which 'force-fed' a sample WHOM source program into the WHOM pipeline so that the backend could be tested without yet having access to the emulation environment provided by WEM. However, we knew that without a testing environment like WEM, where the frontend, backend, and input can be put together into one testing harness, our testing would be severely limited. At this point, we realized that the initial extravagant design of WEM was taking too much time to implement, and took up a minimalist approach. WEM was completed to the bleeding edge of required functionality, and testing commenced using only the emulation environment of WEM. The various tests, such as the original 'force-fed' backend example was saved into the standard test examples as a separate WHOM file to be used for standard testing by loading it through WEM just like any other WHOM source file.

6.2.3 - Multi-Tier Test Cases.

After WEM was able to be used for system testing, we were able to test WHOM programs as a whole, but we also needed a way to do unit testing which isolated individual parts of the system which were either in question due to a bug, or were in process of being implemented. Due to time restrictions, instead of implementing this additional functionality in WEM, we decided to implement some cooperative structure and partitioned our testing based on 'levels' if complexity. Our development code tree has an ever changing set of 'test' programs which are partitioned in levels. The set of WHOM programs designed to test a particular WHOM feature is partitioned into its own level. The details of each level are outlined below.

6.3 - Test Program Listing

The WHOM programs developed for testing are partitioned into tiers of complexity. While the levels are pretty static, the programs within each level continue to grow in number. The details of each level outlined below are a snapshot of the set of programs contained within as of the writing of this document. The number of test programs is expected to grow, especially within the levels of higher complexity.

Level One

The programs in level one are mostly testing the front end of the WHOM pipeline. After we were satisfied with the behavior of the frontend, the level one tier was quickly dropped for more complex functionality testing enclosed in the second level.

whitespace1.whom:

The program is essentially an empty file. This was an attempt to see how the WHOM frontend reacted to empty input.

whitespace2.whom:
Consists of various forms of whitespace, including tabs, spaces, and newline characters. This was an attempt to see how the WHOM frontend reacted to what was a seemingly empty file, but nonetheless contained various control characters.

whitespace_comments1.whom:
The file consists solely of whitespace and single line comments. Attempt to ascertain how the WHOM frontend dealt with single line '//' comments.

whitespace_comments2.whom:
The program is an extensive test of comments. The file consists of single line, multi-line, and nested comments separated by various degrees of whitespace.

Level Two
Level two dealt with testing the parsing and backend logic dealing with declaration and definition of classes, global variables, real-time variables and constructs such as conditional statements and loops.

loop_if_else_test1.whom:
The program tests the *if-else*, *while*, and *for* loops.

once_statement_test1.whom:
The program tests the usage of an expression as the predicate for the *once* statement.

parsing_realtime_vars1.whom:
Simply tests the creation and parsing of real-time variables.

forced_error1.whom:
Tests the WHOM pipeline by forcing an error where an invalid identifier is used as an lvalue.

Level Three
The third level is almost entirely dedicated to testing event logic. Events are a major part of WHOM, and much of our testing concentrated on them.

event_dependency1.whom:
event_dependency2.whom:
These programs test if a global non-real-time number responds to a change in the real-time variable triggered by an event.

event_logic1.whom:
event_logic2.whom:
Test the modification of a real-time variable inside an event triggered by a change in the same real-time variable.

event_logic3.whom:

The program tests the triggering of two executions of the same event.

event_logic4.whom:
Tests event logic by having predicates based on the same class member.

Level Four
This level tests the more complicated features of WHOM and attempts to see how well the various features work together. Programming concepts such as recursion are tested here; alongside with WHOM specific concepts such as dynamically updated real-time variables.

time1.whom:
This example introduces the WHOM real-time 'standard' variables HOUR, MINUTE, and SECOND. These variables dynamically updated by WEM by utilizing current system time. This example tests WEM's ability to deal with this dynamic update by simply defining the existence of the variables themselves.

time2.whom:
This example builds on time1.whom by incorporating an event whereby a global variable changes whenever a change in SECOND occurs. This is to test the functionality of events in WHOM triggered by a time related real-time variable.

recursion1.whom:
This example explores recursion in WHOM by attempting to calculate the Fibonacci numbers.

Level Five
Contains complete WHOM programs. These are the programs that actually 'do something useful'. WHOM programs here attempt to replicate 'real-world' example of the uses of the embedded WHOM system, and are meant for demonstration purposes, rather than testing or debugging.

Experimental Tier
The experimental programs are those that explore the 'theoretically possible' features of WHOM. Things such as polymorphism are attempted here. This level is never part of the standard set of tests which WHOM is expected to pass.

polymorphism_test1.whom:
The program attempts to test polymorphism in WHOM.

Toy Box
Toy box programs are WHOM programs written by individual members of the team while developing their code. These programs are little more than temporary 'hacks' written to test the code currently being worked on. The programs here are not documented, nor are they maintained. If a program here proves to be a valuable as part of the standard test case set, it will be commented, and moved to an appropriate testing tier.

Miscellaneous Tests

Miscellaneous tests are WHOM programs written for either milestone testing purposes, or for testing obscure multi-level spanning things, and which thus do not really belong to any testing tier. These programs are not part of the regular cycle of WHOM 'regression' tests since they are not regularly maintained to keep in sync with the latest WHOM language specifications or parsing functionality.

original_backend_example.whom:
This was the original example hardcoded into the backend class in order to test the backend functionality before WEM was available as a viable testing tool.

include_standard_whom_library.whom:
This code is used simply to test the ability of the WHOM system to 'import' another file into the current source code. More specifically, this program imports the WHOM standard library: *whom.wl*.

milestone2_1.whom:
This code was prepared as a milestone test case. It is a relatively advanced WHOM example, testing such things as the *import* statement, object instantiation, event declaration, and class data member access, among other things. This is one of the first examples to use the WHOM standard library for the purposes of working with 'pre-defined' WHOM classes, such as *Room*, *Door* and *Light*, as well as predefined environment variables such as *TEMPERATURE* and *HOUR*.

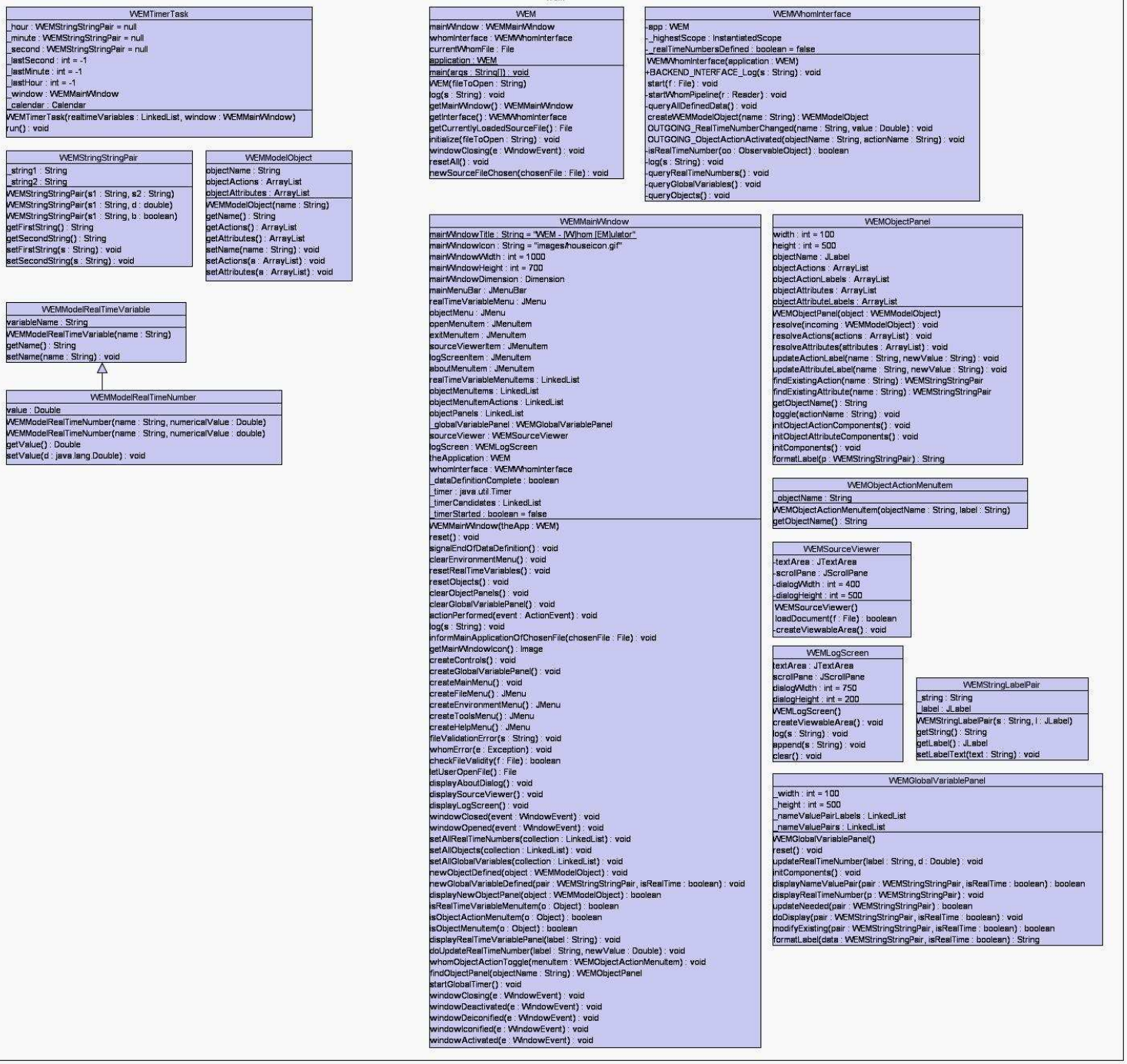
6.4 – WEM Architecture.

6.4.1 – WEM Architecture Overview

The original design of WEM contained an extensive infrastructure complete with data modelers and observers of data which would delegate information and state changes to data representation canvases. This turned out to be much more than we needed. We then undertook a ‘minimalist’ approach to WEM, knowing that what we really needed was simply a tool for testing and eventual demonstration. The diagram below outlines the complete set of WEM classes. Even the diagram however, implies much more complexity that WEM really implements. In a nutshell, WEM is simply an alliance between an application window which is responsible for displaying the current state of the program, as well as allowing the user to simulate changes in various conditions, such as real-time variables, and a WEM/WHOM interface module which holds the responsibility of retrieving information from the backend and delegating the changes of state between WEM and the backend.

6.5 – The complete set of WEM classes.

wem



7.0 Lessons Learned

Rui Kuang (group leader):

Project Management:

Good Lessons:

1. Keep checking of the progress by setting up milestones. These milestones made our project progress fit in our initial schedule perfectly.

2. Good estimate of workload. We restricted the functionality of our language yet powerful, which turns out to be a reasonable setup.

Bad Lessons:

1. Not enough group member communication. Every member is busy and has different schedule, which makes somehow the development of each part independent work.

2. To save effort we cut down our meeting time, which reduced the opportunity for us to learn from each other. I should have managed a better meeting schedule and balanced the effort and the learning time.

Front-end Development:

Good Lessons:

1. Before we started to implement the lexer and parser, we already have clear definition of grammar. If otherwise, we would have needed to do a lot of rework.

2. Implementation of lexer and parser was finished before the implementation of backend fully started. In such a way, we had minimum amount of coordination problem between front-end and back-end

Bad Lessons:

1. Debugging of the front-end grammar and semantics is tricky. We need to cover not only legal grammars and semantics but also catch illegal ones to save the effort of back-end implementation.

Arvid Bessen:

The most important lesson I learned in this project is that laziness would have paid off. All problems I had to encounter and solve were presented only one or two weeks later in

class. Besides this, coding a Whom interpreter was a really nice experience and a good exercise in object- and pattern oriented programming, since both approaches are well suited here. It helped a lot that we had split the project so cleanly into different modules with clearly defined interfaces, but this clean separation also created the problem that other people did not know about the design structure and philosophy of the other parts. Perhaps it is better to fix an interface in advance even if you know that the interface design will be flawed and create a wrapper around this interface, instead of waiting until you are able to create an interface that actually mirrors the innate design.

Another lesson I learned is that design patterns that seemed perfectly natural to apply to me (especially observers), made it harder to understand my code for others.
Finally I still think that the decision to write a high-level interpreter was an unlucky choice. The alternatives (a multi-stage compiler: Whom -> Java code -> Java bytecode, or a direct compiler: Whom -> Java bytecode or assembler) would have been much simpler (Java code) or much more efficient and easier to debug (Java bytecode). The approach we chose combines the disadvantages of the two other approaches without necessarily reaping the benefits.

Andrey Butov: While I am happy that the project exposed me to the field of compiler design and implementation, I must admit that it has made me eager to approach a similar project with a more hands-on approach; meaning building a scanner, a parser, and all other tools from the ground up without relying on compiler creation tools. As a personal project not bounded by time constraints (something outside the scope of an academic or a professional environment), it would be quite a bit more interesting to take such an approach.

As far as this specific project is concerned, I am not nearly as satisfied with the testing process as I would like to have been. I am quite content with the development of WEM, although the initial design was quite over-engineered and required some time to "dumb down" – thus time wasted. The structure of test runs themselves could have been better planned. In the end, while some thought has been put into the various levels and tiers of tests, the test themselves originally came from the mind of an individual developer while working on a bug which spontaneously popped up. I would have preferred a more rigorous approach to testing, requiring more thought to be put into developing the test code itself.

Lana Starshinina: One of the things that I learned while doing the project that it can be very difficult to find time to meet every week. Another lesson was the importance of having agendas for the meetings. That was particularly important because it greatly increased the productivity of our meetings. Also I thought it was great that we set specific milestones and created an overall project plan very early on. The entire project was relatively evenly spread out throughout the semester and at no time we had to be extremely stressed out about finishing a particular part on time.

Another lesson that I learned was that it is better to keep everything relatively simple. At first we perhaps attempted to do too much, so at the end we needed to sacrifice some of the interesting functionality.

8.0 Appendix

8.1 WHOM Grammar

8.1.1 Lexical rules

ALPHA 'a'..'z' | 'A'..'Z' | '_'
DIGIT '0'..'9'
ID ALPHA (ALPHA | DIGIT)*
NUMBER (DIGIT)+('.' (DIGIT)*)?
STRING ""(~("" | '\n') | (""""))*""
WS (' ' | '\t')+
NL ('\n' | ('\r' '\n') => '\r' '\n' | '\r')
COMMENT : ("/*" ((NL)| ~('\n' | '\r'))* "*/" | "//" (~('\n' | '\r'))* (NL))

8.1.2 Syntactic rules

program (import_file)* (rtdef | classdef | declaration | objdec)* (eventimplement)*
import_file "import" STRING (‘,’ STRING)* ‘;’
rtdef "realtime" type (ID | decassgn) (‘,’ (ID | decassgn))* ‘;’
eventimplement "once" (claID | ‘(‘ expression ‘)’) block
declaration type (ID | decassgn) (‘,’ (ID | decassgn))* ‘;’
decassgn ID ‘=‘ expression
objdec ID ID (‘,’ ID)* ‘;’
classdef "class" ID ("extends" ID)? classblock
classblock ‘{‘ (objdec | events | methoddef | declaration)* ‘}’
attributes (declaration|objdec)*
events ID ‘;’
methoddef (type | "void" | ID) ID argsdef block
argsdef ‘(‘ (type ID (‘,’ type ID)*)? ‘)’
statement for_stat | if_stat | while_stat | return_stat | assignment | expression ‘;’ |
dispatch_stat | block
block ‘{‘ (declaration|statement)* ‘}’
for_stat "for" ‘(‘ (assignlist)? ‘;’ expression ‘;’ (assignlist)? ‘)’ statement
if_stat "if" ‘(‘ expression ‘)’ statement
while_stat "while" ‘(‘ expression ‘)’ statement
return_stat "return" (expression)? ‘;’
dispatch_stat "dispatch" (claID|ID) ‘;’
assignment (claID|ID) ‘=‘ expression ‘;’
assignlist (claID|ID) ‘=‘ expression (‘,’ (claID|ID) ‘=‘ expression)*
exp_list (expression (‘,’ expression)*)?
expression cond_term (‘|’ cond_term)*
cond_term cond_factor (‘&’ cond_factor)*

cond_factor (‘!’)? cond_relat
cond_relat exp_arith ((‘>=’|’<=’|’==’|’!=’|’>’|’<’) exp_arith)? | "true" | "false"
exp_arith arith_term ((‘+’ | ‘-‘) arith_term)*
arith_term arith_factor ((‘*’ | ‘/’) arith_factor)*
arith_factor (‘+’ | ‘-‘)? r_value
r_value NUMBER | methodcall | claID | ID | STRING | ‘(‘ expression ‘)’
methodcall (claID|ID) ‘(‘ exp_list ‘)’
claID ID (‘.’ ID)+
type "number"|"string"

8.2 WHOM Source code

- makefile used
- frontend: lexer and parser
- backend
- source for test programs

Makefile:

```
# Makefile
# Here you add all subdirs you want to compile
SUBDIRS = whom wem

# This is just machinery, do not touch
export CLASSPATHROOT = $(shell pwd)

include $(CLASSPATHROOT)/Rules.make

.PHONY: clean doc

clean:
      rm -f `find $(SUBDIRS) -name "*.class"`
      rm -f `find $(SUBDIRS) -name "*.html"`
      rm -rf *.html package-list resources stylesheet.css
      rm -f whom/WHOMParser.java whom/WHOMTokenTypes.txt
whom/WHOMLexer.java whom/WHOMTokenTypes.java whom/WHOMWalker.java

doc:
      javadoc -source 1.4 `find $(SUBDIRS) -name "*.java"`

# Rules.make
JAVAFILES := $(wildcard *.java)

CLASSFILES = $(subst .java,.class,$(JAVAFILES))

HTMLFILES = $(subst .java,.html,$(JAVAFILES))

.PHONY: all subdirs $(SUBDIRS)

all: $(CLASSFILES) $(SUBDIRS)

# all: $(CLASSFILES) $(SUBDIRS) $(HTMLFILES)

$(SUBDIRS):
```

```
	$(MAKE) -C $@ $(TARGET)

%.class: %.java
	javac -deprecation -source 1.4 -classpath
$(CLASSPATHROOT):$(CLASSPATH) $<

%.html: %.java
	javadoc -source 1.4 -classpath $(CLASSPATHROOT):$(CLASSPATH) $<
```

Front-end:

```
/*
 * whom_parse.g : the lexer and the parser of WHOM, in ANTLR grammar.
 * Author Rui Kuang rkuang@cs.columbia.edu
 * Date : 10/16/2003

 * revise history:
   11/13/2003 Rui Kuang, add meaningful names for each token override
                         reportError function to support my error piles
   12/01/2003 Rui KUang, remove "put" and "any", which is decided not to
be in our language
 */

header {
package whom;
}

class WHOMLexer extends Lexer;

options{
    k = 2;
    charVocabulary = '\3'..'\377';
    testLiterals = false;
    exportVocab = WHOM;
}

{
    //error handling class
    WHOMException errhandler = new WHOMException();
    public void setErrorHandler(WHOMException e)
    {
       errhandler.errmsgs = e.errmsgs;
       errhandler.warnings= e.warnings;
       errhandler.runtime = e.runtime;
    }
    public void reportError( String s ) {
        errhandler.adderr(s);
    }
    public void reportError( RecognitionException e ) {
        errhandler.adderr(e);
    }
}

protected
ALPHA   : 'a'..'z' | 'A'..'Z' | '_';

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

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

NL      : ('\n' | ('\r' '\n') => '\r' '\n' | '\r')
```

```
            { $setType(Token.SKIP); newline(); }
        ;

COMMENT : ( "/*" (
                    options {greedy=false;} :
                    (NL)
                    | ~( '\n' | '\r' )
                )* "*/"
            | "//" (~( '\n' | '\r' ))* (NL)
          )
         { $setType(Token.SKIP);
           /*System.out.println("Found comment: "+getText());*/}
        ;

LPAREN  : '(';//{System.out.println("Match LPAREN");};
RPAREN  : ')';//{System.out.println("Match RPAREN");};
MULT    : '*';
PLUS    : '+';
MINUS   : '-';
RDV     : '/';
SEMI    : ';';
LBRACE  : '{';
RBRACE  : '}';
ASGN    : '=';
COMMA   : ',';
GE      : ">=";
LE      : "<=";
GT      : '>';
LT      : '<';
EQ      : "==";
NEQ     : "!=";
AND     : '&';
OR      : '|';
NOT     : '!';
DOT     : '.';

ID  options { testLiterals = true; paraphrase="an identifier";}
        : ALPHA (ALPHA|DIGIT)*
        //{System.out.println("Found identifier: "+getText());}
        ;

/* NUMBER example:
 * 1, 1e, 1., 1.e10, 1.1, 1.1e10
 */

NUMBER options { paraphrase="a number";}
        : (DIGIT)+ ('.' (DIGIT)*)?
        //{System.out.println("Found numeric: "+getText());}
        ;

STRING options {paraphrase="a string";}
        : '"'!
             (  ~('"' | '\n')
              | ('"'!'"')
             )*
          '"'!
         //{System.out.println("Found string: "+getText());}
        ;


//EXIT    : '%' { System.exit(0);};


/*
 Parser starts here!
```

```
*/

class WHOMParser extends Parser;

options{
    k = 2;
    buildAST = true;
    exportVocab = WHOM;
}


tokens {
SIGN_MINUS;
SIGN_PLUS;
CLAID;
ASSIGNLIST;
EXP_LIST;
METHODCALL;
DECLARATION;
RTDECLARATION;
OBJDEC;
RETURNCLASS;
EVENTS;
METHODDEF;
BLOCK;
ARGSDEF;
PROG;
}

{
    //error handling class
    WHOMException errhandler = new WHOMException();
    public void setErrorHandler(WHOMException e)
    {
       errhandler.errmsgs = e.errmsgs;
       errhandler.warnings= e.warnings;
       errhandler.runtime = e.runtime;
    }
    public void reportError( String s ) {
        errhandler.adderr(s);
    }
    public void reportError( RecognitionException e ) {
        errhandler.adderr(e);
    }
}

program
        :(import_file)* (rtdef|classdef|declaration|objdec)*
(eventimplement)*
        {#program = #([PROG,"PROG"],program);}
        ;

import_file
        : "import"^ STRING (COMMA! STRING)* SEMI!
        ;

rtdef
        : "realtime"! type (ID | decassgn) (COMMA! (ID | decassgn))*
SEMI!
        {#rtdef=#([RTDECLARATION,"RTDECLARATION"],rtdef);}
        ;

eventimplement
        :  "once"^ (claID| LPAREN! expression RPAREN!) block
        ;
```

```
declaration
        :type (ID | decassgn) (COMMA! (ID | decassgn))* SEMI!
        {#declaration=#([DECLARATION,"DECLARATION"],declaration);}
        ;

decassgn
        :ID ASGN^ expression
        ;


objdec  :
        ID ID (COMMA! ID)* SEMI!
        {#objdec=#([OBJDEC,"OBJDEC"],objdec);}
        ;

classdef
        :"class"^ ID ("extends" ID)? classblock
        ;


classblock
        : LBRACE! ( (ID ID (COMMA|SEMI)) => objdec
                  | events
                  | ((type|"void") ID LPAREN) => methoddef
                  | declaration)* RBRACE!
        ;

attributes
        :
         (declaration|objdec)*
        ;

methods
        :
        (methoddef)*
        ;

events
        :
        ID SEMI!
        {#events=#([EVENTS,"EVENTS"],events);}
        ;

returnclass
        :
        ID
        {#returnclass=#([RETURNCLASS,"RETURNCLASS"],returnclass);}
        ;

methoddef
        :(type|"void"|returnclass) ID argsdef block
        {#methoddef=#([METHODDEF,"METHODDEF"],methoddef);}
        ;

argsdef
        :LPAREN! (type ID)? (COMMA! type ID)* RPAREN!
        {#argsdef=#([ARGSDEF,"ARGSDEF"],argsdef);}
        ;

statement
        :for_stat
        |if_stat
        |while_stat
        |return_stat
```

```
        |((claID|ID) ASGN) => assignment
        |expression SEMI!
        |dispatch_stat
        | block
        ;

block
        :LBRACE! (declaration|statement)* RBRACE!
        {#block=#([BLOCK,"BLOCK"],block);}
        ;

for_stat
        :"for"^ LPAREN! (assignlist)? SEMI! expression SEMI!
(assignlist)? RPAREN! statement
        ;

if_stat
        :"if"^ LPAREN! expression RPAREN! statement
        (options{greedy=true;}: "else"! statement )?
        ;

while_stat
        :"while"^ LPAREN! expression RPAREN! statement
        ;

return_stat
        :"return"^ (expression)? SEMI!
        ;

dispatch_stat
        :"dispatch"^ (claID|ID) SEMI!
        ;

assignment
        :(claID|ID) ASGN^ expression SEMI!
        ;

assignlist
        :(claID|ID) ASGN^ expression ( COMMA! (claID|ID) ASGN^
expression)*
        {#assignlist = #([ASSIGNLIST,"ASSIGNLIST"],assignlist);}
        ;

exp_list
        : expression (COMMA! expression)*
        {#exp_list = #([EXP_LIST,"EXP_LIST"],exp_list);}
        | //nothing
        {#exp_list = #([EXP_LIST,"EXP_LIST"],exp_list);}
        ;

expression
        : cond_term (OR^ cond_term)*
        ;

cond_term
        : cond_factor (AND^ cond_factor)*
        ;

cond_factor
        : (NOT^)? cond_relat
        ;

cond_relat
        : exp_arith ((GE^|LE^|EQ^|NEQ^|GT^|LT^) exp_arith)?
        | "true" | "false"
```

```
        ;

exp_arith
        : arith_term ((PLUS^ | MINUS^) arith_term)*
          //{System.out.println("exp_arith");}
        ;

arith_term
        : arith_factor ((MULT^ | RDV^) arith_factor)*
        ;

arith_factor
        : ( PLUS! {#arith_factor =
#([SIGN_PLUS,"SIGN_PLUS"],arith_factor);}
          | MINUS! {#arith_factor =
#([SIGN_MINUS,"SIGN_MINUS"],arith_factor);}
          )? r_value
        ;

r_value
        :NUMBER //{System.out.println("match number");}
        |((claID|ID) LPAREN) => methodcall
        | claID
        | ID
        |STRING
        |LPAREN! expression RPAREN! //{System.out.println("Match the
recursion");}
        ;

methodcall
        :(claID|ID) LPAREN! exp_list RPAREN!{#methodcall =
#([METHODCALL,"METHODCALL"],methodcall);}
        ;

claID
        : ID (DOT! ID)+ {#claID = #([CLAID,"CLAID"],claID);}
        ;

type
        : "number"|"string"
        ;
```

AST Walker:

```
//
//  Backend.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
/*
 revise history:
 2003/11/13 Rui Kuang, add error handling part
*/

header {
    package whom;
    import java.util.*;
    import whom.*;
    import whom.backend.*;
}

class WHOMWalker extends TreeParser;
options {
```

```
    importVocab=WHOM;
}

{
    //error handling class
    static WHOMException errhandler = new WHOMException();
    static final String error_sep = " : ";
    public void setErrorHandler(WHOMException e)
    {
        errhandler.errmsgs = e.errmsgs;
        errhandler.warnings= e.warnings;
        errhandler.runtime = e.runtime;
    }
    public void reportError( String s ) {
        errhandler.adderr(s);
    }
    public void reportError( RecognitionException e ) {
        errhandler.adderr(e);
    }
    public void reportError( Exception e, String s ) {
        // System.err.println("excep:"+e);
        if( e instanceof RuntimeException )
        e.printStackTrace();
        errhandler.adderr(e.getMessage()+error_sep+s);
    }

    public static void reportBKError( Exception e, String s ) {
        if( e instanceof RuntimeException )
        e.printStackTrace();
        errhandler.adderr(e.getMessage()+error_sep+s);
    }

    boolean inMethodBlock = false;
}

program
    : #(PROG (importing)*  { // begin block for all initializations
                // we begin a new block...
                whom.Backend.statements.addFirst( new LinkedList() );
                // whom.Backend.beginBlock();
            }
            (progbody)* (eventimplement)* {
                // take all the initialization code and mush it to one
block
                // attention: we do not want to forget the scope
                whom.Backend.enqueueCode(
                    new whom.backend.Block(
                        (LinkedList)
whom.Backend.statements.removeFirst(),
                        ParsingScope.current()
                    )
                );
                whom.backend.Statement s = whom.Backend.popCode();
                // and execute it
                try {
                    s.execute();
                }
                catch(Exception e) {
                    reportError(e, "trying to initialize variables.");
                }
            }
        )
    ;

importing
```

```
    : #("import" (s:STRING {
                    System.out.println("We import the file:
"+s.getText());
                    try {
                        // Try to open the file..
                        java.io.DataInputStream dis = new
java.io.DataInputStream(
                            new java.io.FileInputStream(s.getText())
                        );

                        // Create lexer
                        WHOMLexer lexer = new WHOMLexer(dis);

                        // Create parser
                        WHOMParser parser = new WHOMParser(lexer);

                        // Create walker
                        WHOMWalker walker = new WHOMWalker();

                        // Parse the input expression
                        parser.program();
                        antlr.CommonAST t =
(antlr.CommonAST)parser.getAST();

                        // Print the resulting tree out in LISP notation
                        System.out.println(t.toStringList());

                        // Traverse the tree created by the parser
                        walker.program(t);
                    }
                    catch( java.io.IOException e ) {
                        reportError(e, " could not open, or read:
"+s.getText());
                    }
                    catch(Exception e) {
                        reportError(e, " during parsing
"+s.getText()+"\n");
                    }
                }
            )
        )
    ;

progbody
    : realtime
    | classdef
    | declaration
    | objdec
    ;


realtime
    : #(RTDECLARATION realtimetail)
    ;

realtimetail
{
    whom.backend.Expression expr = null;
}
    : ("number" (x:ID {
                    try {
                        whom.Backend.createRealtimeNumber(x.getText());
                    }
                    catch( Exception e ) {
```

```
                        reportError(e, "Could not create number
"+x.getText());
                    }
                }
            | #(ASGN y:ID expr=expression {
                        if( expr.returntype() != Type.Number &&
expr.returntype()!=Type.Boolean)
                        reportError("Variable initialization error, the
delcaration of "+y.getText()+" ignored.");
                        else
                        {
                              try {

whom.Backend.createRealtimeNumber(y.getText());
                                  whom.Backend.enqueueCode(
                                      new whom.backend.Assignment(
                                          whom.Backend.lu(y.getText()),
expr
                                      )
                                  );
                              }
                              catch (Exception e) {
                                  reportError(e, "trying create the
realtime number "+y.getText());
                              }
                        }
                    }
                )
            )+
        )
    | ("string" (xs:ID {
                    try {
                        whom.Backend.createRealtimeString(xs.getText());
                    }
                    catch( Exception e ) {
                        reportError(e, "Could not create string
"+xs.getText());
                    }
                }
            | #(ASGN ys:ID expr=expression {
                        if( expr.returntype() != Type.String)
                        reportError("Variable initialization error, the
delcaration of "+y.getText()+" ignored.");
                        else
                        {
                              try {

whom.Backend.createRealtimeString(ys.getText());
                                  whom.Backend.enqueueCode(
                                      new whom.backend.Assignment(
                                          whom.Backend.lu(ys.getText()),
expr
                                      )
                                  );
                              }
                              catch (Exception e) {
                                  reportError(e, "trying create the
realtime string "+ys.getText());
                              }
                        }
                    }
                )
            )+
        )
    ;
```

```
classdef
{
    String superclassname = "Root";
}
    : #("class" classname:ID (superclassname=classdeftail)? {
                try {
                    whom.Backend.createClassAndBeginClassBlock(classname
.getText(), superclassname);
                }
                catch(Exception e) {
                    reportError(e, "trying to create class "+classname);
                }
            }
            classbody
        ) {
            try {
                whom.Backend.endClassBlock();
                System.out.println("Created class
"+classname.getText());
            }
            catch(Exception e) {
                reportError(e, "trying to create class "+classname);
            }
            // System.out.println("End of class definition");
        }
    ;

classdeftail returns [String superclassname = "Root"]
    : ("extends" superclassnameid:ID {
                superclassname = superclassnameid.getText();
            }
        )
    ;

classbody
    : (options{greedy=true;}: attributedec | objdecinclass | methoddef |
events)* {
            // System.out.println("Found classbody");
        }
    ;

attributedec
    : #(DECLARATION attributedectail)
    ;


// WORKS ?
attributedectail
{
    whom.backend.Expression expr = null;
}
    : ("number" (x:ID {
                    try {
                        whom.Backend.currentClass().addAttribute(x.getTe
xt(), whom.backend.Type.Number);
                    }
                    catch(Exception e) {
                        reportError(e, "trying to add the number "
                            +x.getText()+" to "
                            +whom.Backend.currentClass().getClassname()
                        );
                    }
                }
            | #(ASGN y:ID expr=expression {
```

```
                        if( expr.returntype() != Type.Number &&
expr.returntype()!=Type.Boolean)
                        reportError("Variable initialization error, the
delcaration of "+y.getText()+" ignored.");
                        else
                        {
                              try {
                                whom.Backend.currentClass().addAttribute
(
                                    y.getText(),
whom.backend.Type.Number
                                );
                                whom.Backend.enqueueCode(
                                    new whom.backend.Assignment(
                                          whom.Backend.lu(y.getText()),
expr
                                    )
                              );
                              }
                              catch(Exception e) {
                                reportError(e, " failed to add the
number "
                                    +y.getText()+" to "
                                    +whom.Backend.currentClass().getClas
sname()
                              );
                              }
                        }
                  }
               )
            )+
         )
      | ("string" (xs:ID {
                  try {
                      whom.Backend.currentClass().addAttribute(xs.getT
ext(), whom.backend.Type.String);
                  }
                  catch(Exception e) {
                      reportError(e, "trying to add the string "
                          +xs.getText()+" to "
                          +whom.Backend.currentClass().getClassname()
                      );
                  }
               }
            | #(ASGN ys:ID expr=expression {
                        if( expr.returntype() != Type.String)
                        reportError("Variable initialization error, the
delcaration of "+y.getText()+" ignored.");
                        else
                        {
                              try {

whom.Backend.currentClass().addAttribute(
                                    ys.getText(),
whom.backend.Type.String
                                 );
                                 whom.Backend.enqueueCode(
                                     new whom.backend.Assignment(
                                         whom.Backend.lu(ys.getText()),
expr
                                     )
                                 );
                              }
                              catch(Exception e) {
```

```
                                    reportError(e, " failed to add the
string "
                                        +y.getText()+" to "

+whom.Backend.currentClass().getClassname()
                                    );
                                }
                            }
                        )
                    )+
                )
            ;

// WORKS ?
objdecinclass
    : #(OBJDEC classname:ID (instancename:ID {
                        try {
                            whom.Backend.currentClass().addAttribute(
                                instancename.getText(),
                                whom.Backend.getClass(classname.getText())
                            );
                            // System.out.println("Found OBJDEC");
                        }
                        catch( Exception e ) {
                            reportError(e, " failed to add "
                                +instancename.getText()+" of class "
                                +classname.getText()+" to "
                                +whom.Backend.currentClass().getClassname()
                            );
                        }
                    }
                )+
            )
        ;

// WORKS ?
methoddef
{
    java.util.ArrayList args;
    int returntype = -1;
    String classname = null;
}
    : #(METHODDEF ("void" {
                        returntype = whom.backend.Type.Void;
                    }
                | "number" {
                        returntype = whom.backend.Type.Number;
                    }
                | "string" {
                        returntype = whom.backend.Type.String;
                    }
                | #(RETURNCLASS classnameid:ID)  {
                        returntype = whom.backend.Type.Object;
                        classname = classnameid.getText();
                    }
                ) methodname:ID {
                    // We will get some parameters soon, so we start a new
scope to
                    // store them somewhere
                    BlockScope bsco = new BlockScope();
                    ParsingScope.setCurrent(bsco);

                } args=argsdef {
                    // Let us create an entry for every parameter
```

```
                // Add all the parameters to the parameter scope
                Iterator it = args.iterator();
                while( it.hasNext() ) {
                    try {
                        String aname = (String) it.next();
                        Attribute attrib = (Attribute) it.next();
                        ParsingScope.current().add(aname, attrib);
                    }
                    catch(Exception e) {
                        reportError(e, " went wrong in attribute
creation ");
                    }
                }
                try {
                    if( returntype != whom.backend.Type.Object ) {
                        whom.Backend.currentClass().addMethod(
                            returntype,
                            methodname.getText(),
                            args,
                            ParsingScope.current()
                        );
                    }
                    else {
                        whom.Backend.currentClass().addMethod(
                            whom.Backend.getClass(classname),
                            methodname.getText(),
                            args,
                            ParsingScope.current()
                        );
                    }
                }
                catch( Exception e ) {
                    reportError(e, "could not create method "
                        +methodname.getText()
                    );
                }
                // now we allow return statements
                inMethodBlock = true;
            } block {
                // now we disallow return statements
                inMethodBlock = false;
                try {
                    Method meth = whom.Backend.currentClass().getMethod(
                        methodname.getText(), args.size() / 2
                    );
                    meth.setBody( (whom.backend.Block)
whom.Backend.popCode() );

                    // remove the parameters
                    ParsingScope.removeCurrent();
                }
                catch( Exception e ) {
                    reportError(e, "could not create method body for "
                        +methodname.getText()
                    );
                }
                // System.out.println("Found methoddef");
            }
        )
    ;

// WORKS ?
argsdef returns [ java.util.ArrayList al = new java.util.ArrayList() ]
    : #(ARGSDEF ( ("number" num:ID) {
                    al.add( num.getText() );
```

```
                    al.add( new
whom.backend.Attribute(whom.backend.Type.Number) );
                | ("string" s:ID)  {
                    al.add( s.getText() );
                    al.add( new
whom.backend.Attribute(whom.backend.Type.String) );
                | #(OBJDEC classname:ID instancename:ID) {
                    try {
                        al.add( s.getText() );
                        al.add( new
whom.backend.Attribute(whom.Backend.getClass(classname.getText()) ) );
                    }
                    catch( Exception e) {
                        reportError(e,"trying to parse
"+instancename.getText()
                            +" as a parameter of "
                            +" type "+classname.getText()
                        );
                    }
                }
            )*
        ) {
            // System.out.println("Found argsdef");
        }
    ;

objdec
    : #(OBJDEC classname:ID instancename:ID) {
            try {
                System.out.println("Creating object "+instancename
                    +" of class "+classname);
                whom.Backend.createObject(instancename.getText(),
classname.getText());
            }
            catch(Exception e) {
                reportError(e, "could not create object "+instancename
                    +" of class "+classname
                );
            }

        }
    ;

// WORKS ?
events
    : #(EVENTS x:ID) {
            try {
                whom.Backend.currentClass().addAttribute(
                    x.getText(), whom.backend.Type.Boolean, true
                );
            }
            catch(Exception e) {
                reportError(e, " could not add event "+x.getText());
            }
            //System.out.println("Found events");
        }
    ;

declaration
    : #(DECLARATION declarationtail)
    ;

declarationtail
{
    whom.backend.Expression expr = null;
```

```
}
    : ("number" (x:ID {
                    try {
                        whom.Backend.createNumber(x.getText());
                    }
                    catch( Exception e ) {
                        reportError(e, "Could not create number
"+x.getText());
                    }
                }
            | #(ASGN y:ID expr=expression {
                        if( expr.returntype() != Type.Number &&
expr.returntype()!=Type.Boolean)
                        reportError("Variable initialization error, the
delcaration of "+y.getText()+" ignored.");
                        else
                        {
                              try {

whom.Backend.createNumber(y.getText());
                                  whom.Backend.enqueueCode(
                                      new whom.backend.Assignment(
                                          whom.Backend.lu(y.getText()),
expr
                                      )
                                  );
                              }
                              catch (Exception e) {
                              reportError(e, "could not declare or
initialize "
                                    +" the number "+y.getText()
                              );
                              }
                        }
                    }
                )
            )+
        )
    | ("string" (xs:ID  {
                    try {
                        whom.Backend.createString(xs.getText());
                    }
                    catch( Exception e ) {
                        reportError(e, "Could not create string
"+x.getText());
                    }
                }
            | ("string" ys:ID expr=expression {
                        if( expr.returntype() != Type.String)
                        reportError("Variable initialization error, the
delcaration of "+y.getText()+" ignored.");
                        else
                        {
                              try {
                              whom.Backend.createString(ys.getText());
                              whom.Backend.enqueueCode(
                                  new whom.backend.Assignment(
                                      whom.Backend.lu(ys.getText()),
expr
                                  )
                              );
                              }
                              catch (Exception e) {
                              reportError(e, "could not declare or
initialize "
```

```
                                    +" the string "+y.getText()
                                );
                                }
                        }
                    }
                )
            )+
        )
    ;

// WORK
eventimplement
{
    whom.backend.Expression expr;
}
    : #("once" ( (expr=expression block {
                        if( expr.returntype() != Type.Number &&
expr.returntype()!=Type.Boolean)
                        reportError("Event condition not evaluatable,
event ignored!");
                        else
                        {
                              try {
                              whom.Backend.createEvent(
                                    new
whom.backend.ConditionOnExpression(expr),
                                    whom.Backend.popCode()
                              );
                              }
                              catch( Exception e ) {
                              reportError(e, "could not create event");
                              }
                        }
                    }
                )
            )
        )
    ;

condition returns [whom.backend.Condition cond = null]
{
    whom.backend.Expression expr1;
    whom.backend.Expression expr2;
}
    : x:ID {
            try {
                cond = new whom.backend.ConditionOnExpression(
                    whom.Backend.lu(x.getText())
                );
            }
            catch(Exception e) {
                reportError(e, "trying to establish a condition on
"+x.getText());
            }
        }
    | expr1=claid {
            // Try if it really is an event
            cond = new whom.backend.ConditionOnExpression(expr1);
            // System.out.println("Found event from object condition, we
have to return something!");
        }
    | cond=logicCondition
    ;

logicCondition returns [whom.backend.Condition cond = null]
```

```
{
    whom.backend.Expression expr1;
    whom.backend.Expression expr2;
}
    : #(GE expr1=expression expr2=expression) {
            if( (expr1.returntype() != Type.Number
                && expr1.returntype() != Type.Boolean)
            || (expr2.returntype() != Type.Number
                && expr2.returntype() != Type.Boolean) ) {
                reportError("Not comparable!");
            }
            else {
                cond = new whom.backend.ComparisonGE( expr1, expr2 );
            }
       }
    | #(LE expr1=expression expr2=expression) {
            if( (expr1.returntype() != Type.Number
                && expr1.returntype() != Type.Boolean)
            || (expr2.returntype() != Type.Number
                && expr2.returntype() != Type.Boolean) ) {
                reportError("Not comparable!");
            }
            else {
                cond = new whom.backend.ComparisonLE( expr1, expr2 );
            }
       }
    | #(GT expr1=expression expr2=expression) {
            if( (expr1.returntype() != Type.Number
                && expr1.returntype() != Type.Boolean)
            || (expr2.returntype() != Type.Number
                && expr2.returntype() != Type.Boolean) ) {
                reportError("Not comparable!");
            }
            else {
                cond = new whom.backend.ComparisonGT( expr1, expr2 );
            }
       }
    | #(LT expr1=expression expr2=expression) {
            if( (expr1.returntype() != Type.Number
                && expr1.returntype() != Type.Boolean)
            || (expr2.returntype() != Type.Number
                && expr2.returntype() != Type.Boolean) ) {
                reportError("Not comparable!");
            }
            else {
                cond = new whom.backend.ComparisonLT( expr1, expr2 );
            }
       }
    | #(EQ expr1=expression expr2=expression) {
            if( (expr1.returntype() != Type.Number
                && expr1.returntype() != Type.Boolean)
            || (expr2.returntype() != Type.Number
                && expr2.returntype() != Type.Boolean) ) {
                reportError("= Not comparable!");
            }
            else {
                cond = new whom.backend.ComparisonEQ( expr1, expr2 );
            }
       }
    | #(NEQ expr1=expression expr2=expression) {
            if( (expr1.returntype() != Type.Number
                && expr1.returntype() != Type.Boolean)
            || (expr2.returntype() != Type.Number
                && expr2.returntype() != Type.Boolean) ) {
                reportError("!= Not comparable!");
```

```
            }
            else {
                cond = new whom.backend.ComparisonNE( expr1, expr2 );
            }
        }
    | #(AND expr1=expression expr2=expression) {
            System.out.println("expr1: "+expr1.returntype()+" expr2:
"+expr2.returntype());
            if( (expr1.returntype() != Type.Number
                && expr1.returntype() != Type.Boolean)
            || (expr2.returntype() != Type.Number
                && expr2.returntype() != Type.Boolean) ) {
                reportError("& Not comparable!");
            }
            else {
                cond = new whom.backend.ConditionAnd( expr1, expr2 );
            }
        }
    | #(OR expr1=expression expr2=expression) {
            if( (expr1.returntype() != Type.Number
                && expr1.returntype() != Type.Boolean)
            || (expr2.returntype() != Type.Number
                && expr2.returntype() != Type.Boolean) ) {
                reportError("| Not comparable!");
            }
            else {
                cond = new whom.backend.ConditionOr( expr1, expr2 );
            }
        }
    | #(NOT expr1=expression) {
            if( expr1.returntype() != Type.Number
            && expr1.returntype() != Type.Boolean ) {
                reportError("! Not comparable!");
            }
            else {
                cond = new whom.backend.ConditionNot( expr1 );
            }
        }
    ;

// WORKS ?
// This is a descending list, something like: foo.bar.baz
// The first and the last are special
// Consider static versus dynamic object member lookup...
claid returns [whom.backend.Expression expr = null]
{
    whom.backend.Lookup olu = null;
    java.util.LinkedList memberll = new LinkedList();
}
    : #(CLAID x:ID {
            try {
                olu = whom.Backend.lu(x.getText());
                if( olu.returntype() != Type.Object )
                throw new Exception("Only objects have members,
"+x.getText()
                    +" is not an object!");
            }
            catch( Exception e ) {
                reportError(e, " trying to locate object members");
            }
        }
        (y:ID {
                // we should also check if all except the last
                // member is an object at compile time
                memberll.add(y.getText());
```

```
                }
            )+
        ) {
            expr = new whom.backend.ObjectMemberLookup(olu, memberll);
            // System.out.println("Found class ID:"+olu+" "+memberll);
        }
    ;

// WORKS ?
expr_list returns [ java.util.LinkedList ll = new java.util.LinkedList()
]
{
    whom.backend.Expression expr;
}
    : #(EXP_LIST (expr=expression {
                    ll.add(expr);
                }
            )*
        ) {
            // System.out.println("Found expression list");
        }
    ;

// WORKS ?
// Consider static versus dynamic object member lookup...
// WORK
// add support for
// room1.foo()
// group1.foo()
methodcall_expr returns [ whom.backend.Expression expr = null ]
{
    String methodname = null;
    String current = null;
    whom.backend.Lookup olu = null;
    java.util.LinkedList memberll = new LinkedList();
    java.util.LinkedList el = null;
}
    : #(METHODCALL ( menm:ID {
                    try {
                        olu = new
whom.backend.ThisLookup(whom.Backend.currentClass());
                        methodname = menm.getText();
                    }
                    catch( Exception e ) {
                        reportError(e, " could not find this (maybe we
are not in an object?)");
                    }
                }
            | #(CLAID x:ID {
                        try {
                            olu = whom.Backend.lu(x.getText());
                        }
                        catch( Exception e ) {
                            reportError(e, " could not find
"+x.getText());
                        }
                    }
                    ( y:ID {
                            // the old one was not the method name
                            if( methodname != null  ) {
                                memberll.add(methodname);
                            }
                            // maybe this is it?
                            methodname = y.getText();
                        }
```

```
                    )+
                )
            ) el=expr_list
        ) {
            // System.out.println(methodname+" "+memberll);
            expr = new whom.backend.Methodcall(
                new whom.backend.ObjectMemberLookup(olu,memberll),
                methodname,
                el
            );
            // System.out.println("Found method call
\""+methodname+"\"");
        }
    ;

block
    : {
            // System.out.println("Beginning of block");
            whom.Backend.beginBlock();
        }
#(BLOCK (declaration | statement)*) {
            // System.out.println("End of block");
            whom.Backend.endBlock();
        }
    ;

statement
    : for_stat
    | if_stat
    | while_stat
    | return_stat
    | methodcall_stat
    | assignment
    | dispatch
    | block
    ;

// WORKS ?
for_stat
{
    whom.backend.Expression expr;
}
    : #("for" assignlist expr=expression assignlist statement) {
            whom.backend.Statement initSt, loopInc, forBody;
            forBody = whom.Backend.popCode();
            loopInc = whom.Backend.popCode();
            initSt  = whom.Backend.popCode();
            whom.Backend.enqueueCode(
                new whom.backend.For(
                    initSt,
                    new whom.backend.ConditionOnExpression(expr),
                    loopInc,
                    forBody
                )
            );
        }
    ;

// WORKS ?
assignlist
    : {
            whom.Backend.beginBlock();
        }
#(ASSIGNLIST (assignment)+){
            whom.Backend.endBlock();
```

```
        }
    ;

// WORKS ?
if_stat
{
    whom.backend.Expression expr;
    boolean elseExists = false;
}
    : #("if" expr=expression statement (statement {
                    elseExists = true;
                }
            )?
        ) {
            if( elseExists ) {
                Statement thenBody, elseBody;
                elseBody = whom.Backend.popCode();
                thenBody = whom.Backend.popCode();
                whom.Backend.enqueueCode(
                    new whom.backend.IfThenElse(
                        new whom.backend.ConditionOnExpression(expr),
                        thenBody,
                        elseBody
                    )
                );
            }
            else {
                Statement thenBody;
                thenBody = whom.Backend.popCode();
                whom.Backend.enqueueCode(
                    new whom.backend.IfThen(
                        new whom.backend.ConditionOnExpression(expr),
                        thenBody
                    )
                );
            }
        }
    ;

// WORKS ?
while_stat
{
    whom.backend.Expression expr;
}
    : #("while" expr=expression statement) {
            whom.backend.Statement stat = whom.Backend.popCode();
            whom.Backend.enqueueCode(
                new whom.backend.While(
                    new whom.backend.ConditionOnExpression(expr),
                    stat
                )
            );
        }
    ;

// Need to have return type consistent check (not done)
return_stat
{
    whom.backend.Expression expr;
}
    : #("return" expr=expression) {
            // System.out.println("Found return statement");
            if( ! inMethodBlock ) {
                reportError("Return not allowed outside of methods!");
            }
```

```
            else {
                whom.Backend.enqueueCode( new whom.backend.Return( expr
) );
            }
        }
    ;

// Drop the return value
methodcall_stat
{
    whom.backend.Expression expr;
}
    : expr=methodcall_expr {
            whom.Backend.enqueueCode( new whom.backend.AssignmentToVoid(
expr ) );
        }
    ;

assignment
{
    whom.backend.Expression expr1, expr2;

}
    : #(ASGN ( (x:ID expr2=expression {
                        try {
                            whom.backend.Lookup lu =
whom.Backend.lu(x.getText());
                            if( lu.returntype() != expr2.returntype() )
{
                                throw new Exception("Assignment of
incompatible types!");
                            }
                            if( ! lu.lvalue() || ! expr2.rvalue() ) {
                                throw new Exception("Left side of
assignment must be lvalue and"
                                    +" right side must be rvalue");
                            }
                            whom.Backend.enqueueCode(
                                new whom.backend.Assignment(
                                    lu,
                                    expr2
                                )
                            );
                        }
                        catch( Exception e ) {
                            reportError(e, " could not create assignment
to "
                                +x.getText()
                            );
                        }
                    }
                ) | (expr1=claid expr2=expression  {
                        try {
                            if( expr1.returntype() != expr2.returntype()
) {
                                throw new Exception("Assignment of
incompatible types!");
                            }
                            if( ! expr1.lvalue() || ! expr2.rvalue() ) {
                                throw new Exception("Left side of
assignment must be lvalue and"
                                    +" right side must be rvalue");
                            }
                            whom.Backend.enqueueCode(
                                new whom.backend.Assignment(
```

```
                                    expr1,
                                    expr2
                                )
                            );
                        }
                        catch( Exception e ) {
                            reportError(e, "could not create
assignment");
                        }
                    }
                )
            )
        )
    ;

dispatch
{
    whom.backend.Expression expr;
}
    : #("dispatch" ( (c:ID {
                        try {
                            expr = whom.Backend.lu( c.getText() );
                            if( expr.returntype() != Type.Boolean ) {
                                throw new Exception();
                            }
                            whom.Backend.enqueueCode(
                                new whom.backend.Dispatch(
                                    expr
                                )
                            );
                        }
                        catch( Exception e ) {
                            reportError(e, "could not dispatch
"+c.getText());
                        }
                    }
                | expr=claid {
                        try {
                            if( expr.returntype() != Type.Boolean ) {
                                throw new Exception();
                            }
                            // Let's try to make an event out of it
                            whom.Backend.enqueueCode(
                                new whom.backend.Dispatch(expr)
                            );
                        }
                        catch( Exception e ) {
                            reportError(e, "could not dispatch
"+c.getText());
                        }
                    }
                )
            )
        )
    ;


expression returns [whom.backend.Expression expr=null]
{
    whom.backend.Expression expr1 = null, expr2 = null;
}
    : #(PLUS expr1=expression expr2=expression) {
            if( expr1.returntype() == Type.Boolean ) {
                expr1 = new BooleanToNumber(expr1);
            }
```

```
            if( expr2.returntype() == Type.Boolean ) {
                expr2 = new BooleanToNumber(expr2);
            }
            if( expr1.returntype() != Type.Number || expr2.returntype()
!= Type.Number ) {
                reportError("Operator \"+\" can only be applied to
numbers!");
            }
            expr = new whom.backend.Addition( expr1, expr2 );
        }
    | #(MINUS expr1=expression expr2=expression) {
            if( expr1.returntype() == Type.Boolean ) {
                expr1 = new BooleanToNumber(expr1);
            }
            if( expr2.returntype() == Type.Boolean ) {
                expr2 = new BooleanToNumber(expr2);
            }
            if( expr1.returntype() != Type.Number || expr2.returntype()
!= Type.Number ) {
                reportError("Operator \"-\" can only be applied to
numbers!");
            }
            expr = new whom.backend.Subtraction( expr1, expr2 );
        }
    | #(MULT expr1=expression expr2=expression) {
            if( expr1.returntype() == Type.Boolean ) {
                expr1 = new BooleanToNumber(expr1);
            }
            if( expr2.returntype() == Type.Boolean ) {
                expr2 = new BooleanToNumber(expr2);
            }
            if( expr1.returntype() != Type.Number || expr2.returntype()
!= Type.Number ) {
                reportError("Operator \"*\" can only be applied to
numbers!");
            }
            expr = new whom.backend.Multiplication( expr1, expr2 );
        }
    | #(RDV expr1=expression expr2=expression) {
            if( expr1.returntype() == Type.Boolean ) {
                expr1 = new BooleanToNumber(expr1);
            }
            if( expr2.returntype() == Type.Boolean ) {
                expr2 = new BooleanToNumber(expr2);
            }
            if( expr1.returntype() != Type.Number || expr2.returntype()
!= Type.Number ) {
                reportError("Operator \"\\\" can only be applied to
numbers!");
            }
            expr = new whom.backend.Division( expr1, expr2 );
        }
    | c:ID {
            try {
                expr = whom.Backend.lu( c.getText() );
            }
            catch( Exception e ) {
                reportError(e, " could not look up "+c.getText());
            }
        }
    | (SIGN_PLUS expr1=expression) {
            if( expr1.returntype() == Type.Boolean ) {
                expr1 = new BooleanToNumber(expr1);
            }
            if( expr1.returntype() != Type.Number) {
```

```
                reportError("Operator \"+\" can only be applied to a
number!");
            }
            expr = expr1;
        }
    | (SIGN_MINUS expr1=expression) {
            if( expr1.returntype() == Type.Boolean ) {
                expr1 = new BooleanToNumber(expr1);
            }
            if( expr1.returntype() != Type.Number) {
                reportError("Operator \"-\" can only be applied to a
number!");
            }
            expr = new whom.backend.Subtraction( new
whom.backend.ConstNumberLookup(0), expr1 );
        }
    | n:NUMBER {
            // System.out.println("Found number
"+Double.parseDouble(n.getText().trim()));
            expr = new
whom.backend.ConstNumberLookup(Double.parseDouble(n.getText().trim()));
        }
    | s:STRING {
            // System.out.println("Found string "+s.getText());
            expr = new whom.backend.ConstStringLookup(s.getText());
        }
    | "true" {
            expr = new whom.backend.ConstBooleanLookup(true);
        }
    | "false" {
            expr = new whom.backend.ConstBooleanLookup(false);
        }
    | expr=methodcall_expr
    | expr=claid
    | expr=logicCondition
    ;
```

Backend:

```
//
//  Addition.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Implements the addition of two numbers
 */
public class Addition extends ArithmeticOp {

    /**
     * Constructor
     */
    public Addition(Expression s1in,
                Expression s2in) {
      super(s1in, s2in);
    }

    /**
     * Actually executes the subtraction of
     * the expressions given in the constructor.
```

```
     */
    public ObservableObject execute( ) throws Exception {
      Number n1 = (Number) s1.execute();
      Number n2 = (Number) s2.execute();
      //System.out.println("Adding "+n1.getNumber()+" and
"+n2.getNumber());
      return n1.add(n2);
    }

}
//
//  ArithmeticOp.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Abstract superclass for the arithmetic operations
 * (+,-,*,/).
 */
public abstract class ArithmeticOp extends Expression {
    /**
     * The two numbers that are to be added, subtracted, etc.
     */
    Expression s1, s2;

    /**
     * Constructor, takes the two numbers that are to be added,
     * subtracted, etc as parameters.
     */
    public ArithmeticOp(Expression s1in,
                Expression s2in) {
      s1 = s1in;
      s2 = s2in;
    }

    /**
     * Returns the dependencies of
     * the expressions given in the constructor.
     */
    public Collection getDependencies() throws Exception {
      Collection ll = s1.getDependencies();
      ll.addAll( s2.getDependencies() );
      return ll;
    }

    /**
     * Returns Type.Number
     */
    public int returntype() {
      return Type.Number;
    }

    /**
     * true
     */
    public boolean rvalue() {
      return true;
    }
```

```
    /**
     * false
     */
    public boolean lvalue() {
      return false;
    }
}
//
//  Assignment.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Assigns two expressions to each other.
 */
public class Assignment extends Statement {
    /**
     * The left-hand side of the assignment.
     */
    Expression lhs;

    /**
     * The right-hand side of the assignment.
     */
    Expression rhs;

    /**
     * Constructor, takes the left-hand side of the assignment and
     * the right-hand side of the assignment.
     */
    public Assignment(Expression lhsin,
                      Expression rhsin) {
      lhs = lhsin;
      rhs = rhsin;
    }

    /**
     * Perform the assignment.
     */
    public ObservableObject execute( ) throws Exception {
      ObservableObject lhsobj = lhs.execute();
      lhsobj.assign( rhs.execute() );
      return lhsobj;
    }

}
//
//  AssignmentToVoid.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * This class can be used to execute expressions
```

```
 * that are not assigned to anything
 */
public class AssignmentToVoid extends Statement {
    /**
     * The expression to be executed.
     */
    Expression rhs;

    /**
     * Constructor, takes the expression to be executed.
     */
    public AssignmentToVoid(Expression rhsin) {
      rhs = rhsin;
    }

    /**
     * Exceutes the expression rhs.
     */
    public ObservableObject execute( ) throws Exception {
      return rhs.execute();
    }

}
//
//  Attribute.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

/**
 * An attribute, a better name would be Type.
 * Just describes what kind of variable something is.
 */
public class Attribute extends java.lang.Object {
    /**
     * The type of the attribute, see class Type.
     */
    int type;

    /**
     * for a ordinary type we have to know, whether it is observable
     */
    boolean observable = false;

    /**
     * for an object we also store its class
     */
    Class classref = null;

    /**
     * Constructor for a basic non-observable type.
     */
    public Attribute(int typein) {
      type = typein;
      observable = true;
    }

    /**
     * Constructor for a basic type.
     */
    public Attribute(int typein, boolean observablein) {
      type = typein;
```

```
  observable = observablein;
}

/**
 * Constructor for an object type of the class passed.
 */
public Attribute(Class classrefin) {
  type = Type.Object;
  classref = classrefin;
}

/**
 * Is this an object or a basic type.
 */
public boolean isObject() {
  return type == Type.Object;
}

/**
 * Is this a basic type?
 */
public boolean isVariable() {
  return type == Type.Boolean
      || type == Type.Number
      || type == Type.String;
}

/**
 * Create the kind of variable that this stands for.
 */
public ObservableObject create() throws Exception {
  ObservableObject o = null;
  switch(type) {
  case Type.Boolean:
      o = new Boolean(observable);
      break;
  case Type.Number:
      o = new Number(observable);
      break;
  case Type.String:
      o = new WhomString(observable);
      break;
  case Type.Object:
      o = classref.createObject();
      break;
  default:
      throw new Exception("Attribute of the wrong type requested!");
  }
  return o;
}

/**
 * Create the kind of basic type that this stands for.
 */
public ObservableObject createVariable() throws Exception {
  ObservableObject oo = null;
  switch(type) {
  case Type.Boolean:
      oo = new Boolean(observable);
      break;
  case Type.Number:
      oo = new Number(observable);
      break;
  case Type.String:
      oo = new WhomString(observable);
```

```
            break;
          case Type.Object:
          default:
              throw new Exception("Attribute of the wrong type requested!");
          }
          return oo;
        }

        /**
         * Create the kind of objectthat this stands for.
         */
        public whom.backend.Object createObject() throws Exception {
          whom.backend.Object o = null;
          switch(type) {
          case Type.Object:
              o = classref.createObject();
              break;
          case Type.Boolean:
          case Type.Number:
          case Type.String:
          default:
              throw new Exception("Attribute of the wrong type requested!");
          }
          return o;
        }

        /**
         * The type.
         */
        public int getType() {
          return type;
        }

        /**
         * The class.
         */
        public Class getClassref() {
          return classref;
        }
}
//
//  Backend.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom;

import java.util.*;
import whom.backend.*;
import wem.*;

/**
 * A module that encapsulates the backend of whom.
 */
public class Backend {
    /**
     * All generated code.
     *
     * Here we store all our code
     * This is (you will not believe it) a stack of linked lists
(queues).
     *
     * Yes, this is a bit of an overkill, but
```

```
 * it is necessary for the block structure.
 * Every time a new block is generated, we create a new list on
 * the stack for the new block.
 * When the block finishes the whole list is popped and transformed
 * into one entry in the now current list.
 */
static LinkedList statements;

/**
 * The interface to WEM.
 */
public static WEMWhomInterface wemInterface;

static {
  statements = new LinkedList();
  LinkedList ll = new LinkedList();
  statements.addFirst(ll);
}

/**
 * A stack for intermediate expression buildup.
 *
 * To parse complicated expression we use this stack.
 */
static LinkedList expressions = new LinkedList();

/**
 * These are all classes.
 */
static HashMap classes = new HashMap();

/**
 * Here we store all events.
 *
 * This is just to prevent garbage collection...
 */
static ArrayList events = new ArrayList();

/**
 * This stores all conditions, organized as a stack.
 */
static LinkedList conditions = new LinkedList();

/**
 * Initialization, must be called before Backend is used.
 */
public static void init() {
  // Reset the current state...maybe this isn't the first run.
  reset ( );

  // The scope list terminator
  EmptyScope empty = new whom.backend.EmptyScope();
    empty.instantiate();

  // The scope for everything that follows
  GlobalScope global = new whom.backend.GlobalScope();

    // Make global scope visible everywhere
    ParsingScope.setCurrent(global);
  // should be done at the end of parsing, but it does not matter
  global.instantiate();

    // add some constants
  //global.add( "true", new whom.backend.True() );
  // global.add( "false", new whom.backend.False() );
```

```
      // global variables that are always available (temperature, etc.)

      // The root of all class hierarchies
      classes.put( "Root", new whom.backend.RootClass() );

      // We are not handling any class right now
      currentClass = null;
    }

    /** Reset the current state. */
    private static void reset() {
      ParsingScope.clearScopes();

      InstantiatedScope.clearScopes();

      if ( classes != null )
          classes.clear();

    }

    /** Send a message to the WEM log screen. */
    public static void log(String s) {
      if ( wemInterface != null )
          wemInterface.BACKEND_INTERFACE_Log(s);
    }

    /**
     * Creates a realtime number in the current scope.
     */
    public static void createRealtimeNumber( String name ) throws
Exception {
      //System.out.println("Adding "+name+" to "+current());
      ParsingScope.current().add( name, new
whom.backend.Attribute(Type.Number, true) );
    }

    /**
     * Creates a number in the current scope.
     */
    public static void createNumber( String name )  throws Exception {
      ParsingScope.current().add( name, new
whom.backend.Attribute(Type.Number, false) );
    }

    /**
     * Creates a realtime string in the current scope.
     */
    public static void createRealtimeString( String name )  throws
Exception {
      //System.out.println("Adding "+name+" to "+current());
      ParsingScope.current().add( name, new
whom.backend.Attribute(Type.String, true) );
    }

    /**
     * Creates a string in the current scope.
     */
    public static void createString( String name )  throws Exception {
      ParsingScope.current().add( name, new
whom.backend.Attribute(Type.String, false) );
    }

    /**
     * This will return the variable with this name.
```

```
     */
    public static ObservableObject var(String name) {
      return InstantiatedScope.current().getVariable(name);
    }

    /**
     * This will return the variable with this name casted to a number.
     */
    public static whom.backend.Number num(String name) {
      return (whom.backend.Number)
InstantiatedScope.current().getVariable(name);
    }

    /**
     * This will return the variable with this name casted to a Boolean.
     */
    public static whom.backend.Boolean bool(String name) {
      return (whom.backend.Boolean)
InstantiatedScope.current().getVariable(name);
    }

    /**
     * This will return the object with this name.
     */
    public static whom.backend.Object obj(String name) {
      return InstantiatedScope.current().getObject(name);
    }

    /**
     * This returns a Lookup object which can be used
     * in code to get a variable.
     */
    public static Lookup lu(String name) throws Exception {
      // System.out.println("Looking up "+name+" in
"+ParsingScope.current());
      return ParsingScope.current().lu( name );
    }

    /**
     * Creates a new class with given superclass.
     * Starts the definition of all the entries of a class.
     */
    public static void createClassAndBeginClassBlock(String name, String
superclassname) throws Exception {

      // Find the superclass
      whom.backend.Class superclass = getClass(superclassname);

      ClassScope classScope = null;
      if( superclass != null ) {
          classScope = new ClassScope();

          // make this the active scope for everything that is in the
class definition
          ParsingScope.setCurrent(classScope);
      }

      // this is the "constructor", i.e. what gets executed
      // when a class gets instantiated
      // we begin a new block...
      beginBlock(); //statements.addFirst( new LinkedList() );

      // We create a new Class object
      currentClass = new whom.backend.Class(name, superclass, classScope
);
```

```
      // Register the class
      classes.put( name, currentClass );
    }

    /**
     * Ends the definition of all the entries of a class.
     */
    public static void endClassBlock() throws Exception {
      // end constructor block
      endBlock();

      // get the instructions for the instructor...
      Block constructorBlock = (Block) popCode();

      // create a constructor from that
      currentClass.createConstructor(constructorBlock);

      // forget the scope of the class
      ParsingScope sco = (ParsingScope) ParsingScope.removeCurrent();

      // ... and create a new scope for everything that follows
      GlobalScope global = new GlobalScope();
      ParsingScope.setCurrent(global);
      global.instantiate();

      currentClass = null;
    }

    /**
     * The class we are currently building.
     */
    static whom.backend.Class currentClass;

    /**
     * Get the class we are currently building.
     */
    public static whom.backend.Class currentClass() {
      return currentClass;
    }

    /**
     * Creates a new object with given class name.
     */
    public static void createObject(String name, String classname)
throws Exception {
      ParsingScope.current().add( name, new
Attribute(getClass(classname)) );
      // System.out.println("We created:"+obj(name));
    }

    /**
     * Returns a reference to the Class object for a class with this
name.
     */
    public static whom.backend.Class getClass(String name) throws
Exception {
      whom.backend.Class cl = (whom.backend.Class) classes.get(name);
      if( cl == null )
          throw new Exception("Could not find class with name "+name);
      return cl;
    }

    /**
     * Starts a block.
```

```
     *
     * This function starts a new scope. After the block is finished
     * you have to call endBlock() to build the block.
     */
    public static void beginBlock() {
      // System.out.println("Starting block : "+ParsingScope.scopes);

      // everything that is defined in the block goes in an own scope
      BlockScope bsco =new BlockScope();
      ParsingScope.setCurrent(bsco);

      // we begin a new block...
      statements.addFirst( new LinkedList() );
    }

    /**
     * Ends a block.
     *
     * When this function is called the old scope reigns again and
     * have a new instruction on our code list that will execute the
     * whole block
     */
    public static void endBlock() {
      // forget the block's scope and save it with the block...
      enqueueCode( new whom.backend.Block( (LinkedList)
statements.removeFirst(),
                                    ParsingScope.removeCurrent() ) );
      // System.out.println("Ending block : "+ParsingScope.scopes);
    }

    /**
     * Enqueues (adding to the front) the passed statement into the
     * list of statements.
     */
    public static void enqueueCode(Statement cb) {
      LinkedList ll = (LinkedList) statements.getFirst();
      ll.addLast(cb);
    }

    /**
     * Dequeues (removing from the front) a statement
     * from the list of statements.
     */
    public static whom.backend.Statement dequeueCode() {
      LinkedList ll = (LinkedList) statements.getFirst();
      return (whom.backend.Statement) ll.removeFirst();
    }

    /**
     * Pops (removes from the end) a statement from the list of
statements.
     */
    public static whom.backend.Statement popCode() {
      LinkedList ll = (LinkedList) statements.getFirst();
      return (whom.backend.Statement) ll.removeLast();
    }

    /**
     * Creates an event that listens to the condition passed
     * and executes the body passed whenever the updated condition
evaluates
     * to true.
     */
    public static void createEvent( whom.backend.Condition condin,
Statement bodyin ) throws Exception {
```

```
      events.add( new Event(condin, bodyin) );
    }

}
//
//  Block.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * A block is one or more instructions which begins a new scope.
 */
public class Block extends whom.backend.Statement {
    /**
     * The statements in the order they should be executed.
     */
    List statements;

    /**
     * The scope for the block.
     */
    ParsingScope scope;

    /**
     * Constructor, takes the list of statements and the scope of the
block.
     */
    public Block(List statementsin, ParsingScope scopein) {
      statements = statementsin;
      scope = scopein;
    }

    /**
     * Instantiates the scope, makes it current, executes the statements
     * int the order they appear in the list, and removes the scope out
of sight again.
     */
    public ObservableObject execute( ) throws Exception {
      // This block may have some new variables...
      // Because we can be invoked recursively, we have to make sure
      // that we create copies of all variables
      scope.instantiate();

      // System.out.println("Variables visible in this scope:
"+scope.allNames());

      ObservableObject oo = null;

      Iterator it = statements.iterator();
      while( it.hasNext() ) {
          Statement stat = (Statement) it.next();
          try {
            oo = stat.execute();
          }
          catch( ReturnException e ) {
            // clean up
            InstantiatedScope.removeCurrent();
            // and exit
            throw e;
```

```
            }
        }

        InstantiatedScope.removeCurrent();

        return oo;
    }

    /**
     * A block cannot appear in an event condition.
     */
    public LinkedList getDependencies() {
        // this will never appear in an event condition
        return null;
    }
}
//
//  BlockLookup.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Takes a name and returns a reference from a BlockScope.
 */
public class BlockLookup extends Lookup {

    /**
     * Constructor, takes the name to be looked up as parameter.
     */
    public BlockLookup(String namein) {
        super(namein);
    }

    /**
     * Performs the lookup in the current instantiated scope.
     */
    public ObservableObject execute( ) {
        InstantiatedScope isi = InstantiatedScope.current();
        ObservableObject oo = isi.get(name);
        //System.out.println("Dynamic lookup for "+name+" yielded: "+oo);
        return oo;
    }

    /**
     * The return type.
     */
    public int returntype() {
        return returnAttribute().getType();
    }

    /**
     * The return Attribute.
     */
    public Attribute returnAttribute() {
        return ParsingScope.current().getAttribute(name);
    }

    /**
     * True.
```

```
     */
    public boolean rvalue() {
      return true;
    }

    /**
     * True.
     */
    public boolean lvalue() {
      return true;
    }
}
//
//  BlockScope.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * The scope for a block.
 *
 * A scope contains the references to all variables in a block.
 * The variables are indexed by name.
 *
 * We have nested scopes. If a query is not succesful i this scope,
 * we query the surrounding block. Thus a variable in this scope can
 * shadow a variable in the surrounding scope with the same name.
 */
public class BlockScope extends ParsingScope {

    /**
     * Returns an Lookup object for the variable
     * with name name.
     */
    public Lookup objLu(String name) {
      return new BlockLookup(name);
    }

    /**
     * Makes this scope the current scope.
     *
     * May perform additional things to do this.
     * Makes a copy of this scope and all its variables.
     *
     */
    public void instantiate() throws Exception {
      InstantiatedScope.setCurrent( new BlockScopeInstantiated(this) );
    }

}
//
//  BlockScopeInstantiated.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;
```

```
/**
 * The scope for a block.
 *
 * A scope contains the references to all variables in a block.
 * The variables are indexed by name.
 *
 * We have nested scopes. If a query is not succesful i this scope,
 * we query the surrounding block. Thus a variable in this scope can
 * shadow a variable in the surrounding scope with the same name.
 */
public class BlockScopeInstantiated extends InstantiatedScope {

    /**
     * Constructor, takes the surrounding scope as argument.
     */
    public BlockScopeInstantiated(BlockScope bsco) throws Exception {
      Iterator it = bsco.vars.keySet().iterator();
      while( it.hasNext() ) {
          String attrname = (String) it.next();
          Attribute attr = (Attribute) bsco.vars.get(attrname);
          vars.put( attrname, attr.create() );
      }
    }

}
//
//  Boolean.java
//  whom
//
//  Created by Arvid Bessen on Fri Oct 03 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * A class for Events, but the name stuck.
 */
public class Boolean extends ObservableObject
    implements LogicValue, DependencyTracker {

    /**
     * The state of the event.
     */
    boolean value = false;

    /**
     * Constructor, false, not observable
     */
    public Boolean() {

    }

    /**
     * Constructor, false
     */
    public Boolean( boolean observablein ) {
      super(observablein);
    }

    /**
     * Constructor, create a copy of the passed Boolean.
     */
```

```
public Boolean( Boolean boo ) {
  super(boo.observable);
  value = boo.value;
}

/**
 * Boolean
 */
public int getType() {
  return Type.Boolean;
}

/**
 * Is this true or false?
 */
public boolean evaluate() {
    return value;
}

/**
 * Set the value of this and notify the observers if changed.
 */
public void setValue(boolean valuein) {
  if( valuein != value ) {
      value = valuein;
      setChanged();
      notifyObserversLater(this);
  }
}

/**
 * Set the value to true and notifiy the observers unconditionally.
 */
public void dispatch() {
  value = true;
  setChanged();
  notifyObserversLater(this);
}

/**
 * The current value of this.
 */
public boolean getValue() {
  return value;
}

/**
 * Assign another Boolean to this.
 */
ObservableObject assign( ObservableObject oo ) throws Exception {
  Boolean boo = (Boolean) oo;
  setValue( boo.getValue() );
  return this;
}

/**
 * Returns the value of this.
 */
public boolean test() {
  return value;
}

/**
 * Returns the opposite of the value of this.
 */
```

```
    public boolean not() {
      return !value;
    }

    /**
     * Are this object and the other object "true"?
     */
    public boolean and(LogicValue lv) {
      return (value && lv.test());
    }

    /**
     * Are this object or the other object "true"?
     */
    public boolean or(LogicValue lv) {
      return (value || lv.test());
    }

    /**
     * This, if this is observable.
     */
    public Collection getDependencies() {
      LinkedList ll = new LinkedList();
      ll.add(this);
      return ll;
    }
}
//
//  BooleanToNumber.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * The abstract base class for all expressions in whom.
 *
 * Main features: must have a method execute and should be able
 * to tell what their return type is, and whether it is an
 * rvalue and lvalue.
 */
public class BooleanToNumber extends Expression {

    /**
     * The expression that should be a number but is a Boolean
     */
    Expression expr;

    /**
     * Constructor, takes an expression that yields a Boolean
     */
    public BooleanToNumber(Expression exprin) {
      expr = exprin;
    }

    /**
     * The method that is used to execute / evaluate an
     * expression. This must be implemented by the subclassed.
     */
    public ObservableObject execute( ) throws Exception {
      Boolean boo = (Boolean) expr.execute();
```

```
      if( boo.evaluate() )
          return new Number(1);
      else
          return new Number(0);
    }

    /**
     * Type.Number
     */
    public int returntype() {
      return Type.Number;
    }

    /**
     * True.
     */
    public boolean rvalue() {
      return true;
    }

    /**
     * False.
     */
    public boolean lvalue() {
      return false;
    }

    /**
     * The dependencies of expr.
     */
    public Collection getDependencies() throws Exception {
      return expr.getDependencies();
    }

}
//
//  Class.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * A class to represent classes in Whom.
 */
public class Class extends java.lang.Object {
    /**
     * The methods of the class.
     */
    ArrayList methods;

    /**
     * The name of the class.
     */
    String classname;

    /**
     * A reference to the (Whom) super class.
     */
    Class superclass;
```

```
    /**
     * Here we have all our variables.
     *
     */
    ClassScope classScope;

    /**
     * Constructor for a class with name name, super class superclassin,
     * and variable as they can be found in classScopein.
     */
    public Class(String name, Class superclassin, ClassScope
classScopein) {
      classname = name;
      superclass = superclassin;
      classScope = classScopein;
      classScope.setClassRef(this);
      methods = new ArrayList();
    }

    /**
     * The names of all member attributes of the class.
     */
    public Collection lookupAllNames() {
      return classScope.lookupAllNames();
    }

    /**
     * Get the attribute entry for the attribute with the given name.
     */
    public Attribute lookupAttribute(String name) {
      return classScope.lookupAttribute(name);
    }

    /**
     * The class name.
     */
    public String getClassname() {
      return classname;
    }

    /**
     * The classes scope.
     */
    public Scope getClassScope() {
      return classScope;
    }

    /**
     * Creates a constructor that executes the block passed upon
creation
     * of a new object.
     */
    public void createConstructor(Block constructorBlock) throws
Exception {
      //System.out.println("Creating constructor with code
"+constructorBlock);
      addMethod(Type.Void,
              "$init",
              new ArrayList(),
              new EmptyScope()
              );
      getMethod("$init",0).setBody(constructorBlock);
    }

    /**
```

```
     * Add a member of a basic type to this class.
     */
    public void addAttribute(String name, int type) throws Exception {
      Attribute a = new Attribute(type);
      classScope.add( name, a );
    }

    /**
     * Add a perhaps observable member of a basic type to this class.
     */
    public void addAttribute(String name, int type, boolean observable)
throws Exception {
      Attribute a = new Attribute(type, observable);
      classScope.add( name, a );
    }

    /**
     * Add a member of another class to this class.
     */
    public void addAttribute(String name, Class classrefin) throws
Exception {
      Attribute a = new Attribute(classrefin);
      classScope.add( name,  a );
    }

    /**
     * Add a new method to this class that returns a basic type.
     */
    public void addMethod( int returnType,
                    String name,
                    ArrayList parameters,
                    ParsingScope parameterScope ) throws Exception {
      Method m = new Method( returnType, name, parameters,
parameterScope );
      methods.add(m);
    }

    /**
     * Add a new method to this class, which returns an object.
     */
    public void addMethod( Class classname,
                    String name,
                    ArrayList parameters,
                    ParsingScope parameterScope ) throws Exception {
      Method m = new Method( classname, name, parameters, parameterScope
);
      methods.add(m);
    }

    /**
     * Used in instantiating this class by creating all the attributes.
     */
    protected void addAttributesToObjectScope(ObjectScope os) throws
Exception {
      // System.out.println("Adding class "+classname+" to "+os);
      superclass.addAttributesToObjectScope(os);
      // add attributes
      Iterator it = classScope.lookupAllNames().iterator();
      while( it.hasNext() ) {
          String attribname = (String) it.next();
          Attribute attrib = (Attribute)
classScope.lookupAttribute(attribname);
          os.add( attribname, attrib );
      }
    }
```

```
/**
 * Calls the constructor of the super class and afterwards
 * the constructor of this class.
 */
protected void invokeConstructor(Object o) throws Exception {
  //System.out.println("invoking constructor for class "+classname);

  // Call the constructor of the superclass
  if( superclass != null )
      superclass.invokeConstructor(o);

  // execute the constructor
  o.invokeMethod("$init", new LinkedList());
}

/**
 * Create an instantiation of this class.
 */
public Object createObject() throws Exception {
  //System.out.println("Creating object of class "+classname);
  //try { throw new Exception(); }
  //catch(Exception e) { e.printStackTrace(); }
  Object o = new Object( this );

  // Execute constructor
  invokeConstructor(o);

  return o;
}

/**
 * The type of the attribute with this name.
 */
public Attribute attributeType(String attribname) {
  return classScope.lookupAttribute(attribname);
}

/**
 * The return type of the method with this name.
 */
public int methodReturnType(String methodname) {
  Iterator it = methods.iterator();
  while( it.hasNext() ) {
      Method meth = (Method) it.next();
      if( meth.getName().equals(methodname) )
        return meth.returnType();
  }
  // Look in superclass
  return superclass.methodReturnType(methodname);
}

/**
 * The return attribute of the method with this name.
 */
public Attribute methodReturnAttribute(String methodname) {
  Iterator it = methods.iterator();
  while( it.hasNext() ) {
      Method meth = (Method) it.next();
      if( meth.getName().equals(methodname) )
        return meth.returnAttribute();
  }
  // Look in superclass
  return superclass.methodReturnAttribute(methodname);
}
```

```
    /**
     * Get the method with the given name and number of parameters.
     */
    public Method getMethod( String methodname,
                         int numberOfParameters ) throws Exception {
      Iterator it = methods.iterator();
      while( it.hasNext() ) {
          Method meth = (Method) it.next();
          if( meth.getName().equals(methodname)
            && (meth.getNumberOfParams() == numberOfParameters) ) {
            return meth;
          }
      }
      // Look in superclass
      return superclass.getMethod( methodname, numberOfParameters );

    }

    /**
     * Execute the method with name methodname on the passed Object.
     */
    public ObservableObject invoke( Object thisref,
                            String methodname,
                            List parameters ) throws Exception {
      Method meth = getMethod(methodname, parameters.size());
      return meth.invoke(thisref, parameters );
    }
}
//
//  ClassScope.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * The scope for a class.
 *
 * A scope contains the references to all variables in a block.
 * The variables are indexed by name.
 *
 * We have nested scopes. If a query is not succesful in this
class/scope,
 * we query the surrounding class/scope. Thus a variable in this scope
can
 * shadow a variable in the surrounding scope with the same name.
 *
 * A ClassScope is only used during compiling a WHOM program.
 */
public class ClassScope extends ParsingScope {

    /**
     * The class for which this is the scope.
     */
    Class classref = null;

    /**
     * Constructor.
     */
    public ClassScope() {
```

```
    }

    /**
     * Sets the class for which this is the ClassScope.
     */
    public void setClassRef(Class classrefin) {
      classref = classrefin;
    }

    /**
     * Returns true, if the scope (or the enclosing scopes) contains
     * something of this name.
     */
    public boolean has( String name ) {
      if( super.has(name) )
          return true;
      else
          return classref.superclass == null ? false
            : classref.superclass.classScope.has(name);
    }

    /**
     * Returns true, if the scope (or the enclosing scopes) contain
     * a basic type (Number or String) with this name.
     */
    public boolean hasVariable( String name ) {
      if( super.hasVariable(name) )
          return true;
      else
          return classref.superclass == null ? false
            : classref.superclass.classScope.hasVariable(name);
    }

    /**
     * Returns true, if the scope (or the enclosing scopes) contains
     * am object of this name.
     */
    public boolean hasObject( String name ) {
      if( super.hasObject(name) )
          return true;
      else
          return classref.superclass == null ? false
            : classref.superclass.classScope.hasObject(name);
    }

    /**
     * Returns an Lookup object for the variable
     * with name name.
     */
    public Lookup objLu(String name) {

      // First let's get the "this" reference
      ThisLookup tlu = new ThisLookup(classref);

      // Now let's get the members
      LinkedList ll = new LinkedList();
      ll.addFirst(name);
      return new ObjectMemberLookup(tlu, ll);
    }

    /**
     * Add a variable to this scope. Throws an exception if already
present.
     */
```

```
    public void add(String name, Attribute attr) throws Exception {
      if( this.has( name ) )
          throw new Exception("Trying to redefine attribute in same or
subclass!");
      else
          super.add(name, attr);
    }

    /**
     * Returns all names of the variables visible in this scope.
Implementation
     */
    public Collection lookupAllNames() {
      Collection coll = classref.superclass == null ? new LinkedList()
          : classref.superclass.classScope.lookupAllNames();
      coll.addAll( vars.keySet() );
      return coll;
    }

    /**
     * Returns the Attribute of the variable with this name in this
scope.
     */
    public Attribute lookupAttribute(String name) {
      if( this.has(name) )
          return (Attribute) vars.get(name);
      else
          if( classref.superclass == null )
            return new Attribute(Type.Unknown);
          else
            return classref.superclass.classScope.lookupAttribute(name);
    }

    /**
     * Returns all attributes of the class.
     */
    public Collection allAttributes() {
      Collection coll = classref.superclass == null ? new LinkedList()
          : classref.superclass.classScope.allAttributes();
      coll.addAll( vars.values() );
      return coll;
    }

    /**
     * Makes this scope the current instantiated scope.
     *
     * May perform additional things to do this.
     * Should be overridden by subclasses that need this.
     */
    public void instantiate() throws Exception {
      InstantiatedScope.setCurrent( new ObjectScope(this) );
    }
}
//
//  Comparable.java
//  whom
//
//  Created by Arvid Bessen on Fri Oct 03 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
```

```
 * An interface for objects that are comparable.
 */
public interface Comparable {
    /**
     * Returns true if this is less than o
     */
    public boolean lt( Comparable o );

    /**
     * Returns true if this is less or equal than o
     */
    public boolean le( Comparable o );

    /**
     * Returns true if this is equal to o
     */
    public boolean eq( Comparable o );

    /**
     * Returns true if this is not equal to o
     */
    public boolean ne( Comparable o );

    /**
     * Returns true if this is greater than o
     */
    public boolean gt( Comparable o );

    /**
     * Returns true if this is greater or equal than o
     */
    public boolean ge( Comparable o );
}
//
//  Comparison.java
//  whom
//
//  Created by Arvid Bessen on Fri Oct 03 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Abstract superclass that compares two
 * expressions and returns a boolean value.
 */
public abstract class Comparison extends Condition {

    /**
     * The two expressions to be compared.
     */
    Expression cb1, cb2;

    /**
     * Constructor, takes the two expressions to be compared as input.
     */
    public Comparison( Expression cb1in, Expression cb2in ) {
      if( cb1in.returntype() == Type.Boolean )
          cb1in = new BooleanToNumber(cb1in);
      if( cb2in.returntype() == Type.Boolean )
          cb2in = new BooleanToNumber(cb2in);
      cb1 = cb1in;
      cb2 = cb2in;
```

```
    }

    /**
     * Returns the dependencies of the two expressions that are to be
     * compared.
     */
    public Collection getDependencies() throws Exception {
      Collection ll = cb1.getDependencies();
      ll.addAll( cb2.getDependencies() );
      return ll;
    }
}
//
//  ComparisonEQ.java
//  whom
//
//  Created by Arvid Bessen on Fri Oct 03 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Implements the comparison of equality.
 */
public class ComparisonEQ extends Comparison {

    /**
     * Constructor, takes the two expressions to be compared as input.
     */
    public ComparisonEQ( Expression cb1in, Expression cb2in ) {
      super( cb1in, cb2in );
    }

    /**
     * Evaluates the comparision and returns the result.
     */
    public boolean evaluate() throws Exception {
        Comparable n1 = (Comparable) cb1.execute();
        Comparable n2 = (Comparable) cb2.execute();
      return n1.eq(n2);
    }

}
//
//  ComparisonGE.java
//  whom
//
//  Created by Arvid Bessen on Fri Oct 03 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Implements the comparison greater or equal.
 */
public class ComparisonGE extends Comparison {

    /**
     * Constructor, takes the two expressions to be compared as input.
     */
    public ComparisonGE( Expression cb1in, Expression cb2in ) {
```

```
      super( cb1in, cb2in );
    }

    /**
     * Evaluates the comparision and returns the result.
     */
    public boolean evaluate() throws Exception {
        Comparable n1 = (Comparable) cb1.execute();
        Comparable n2 = (Comparable) cb2.execute();
      return n1.ge(n2);
    }

}
//
//  ComparisonGT.java
//  whom
//
//  Created by Arvid Bessen on Fri Oct 03 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Implements the comparison greater than.
 */
public class ComparisonGT extends Comparison {

    /**
     * Constructor, takes the two expressions to be compared as input.
     */
    public ComparisonGT( Expression cb1in, Expression cb2in ) {
      super( cb1in, cb2in );
    }

    /**
     * Evaluates the comparision and returns the result.
     */
    public boolean evaluate() throws Exception {
        Comparable n1 = (Comparable) cb1.execute();
        Comparable n2 = (Comparable) cb2.execute();
      return n1.gt(n2);
    }

}
/
//  ComparisonLE.java
//  whom
//
//  Created by Arvid Bessen on Fri Oct 03 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Implements the comparison less or equal.
 */
public class ComparisonLE extends Comparison {

    /**
     * Constructor, takes the two expressions to be compared as input.
     */
```

```
    public ComparisonLE( Expression cb1in, Expression cb2in ) {
      super( cb1in, cb2in );
    }

    /**
     * Evaluates the comparision and returns the result.
     */
    public boolean evaluate() throws Exception {
        Comparable n1 = (Comparable) cb1.execute();
        Comparable n2 = (Comparable) cb2.execute();
      return n1.le(n2);
    }

}
//
//  ComparisonLT.java
//  whom
//
//  Created by Arvid Bessen on Fri Oct 03 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Implements the comparison less than.
 */
public class ComparisonLT extends Comparison {

    /**
     * Constructor, takes the two expressions to be compared as input.
     */
    public ComparisonLT( Expression cb1in, Expression cb2in ) {
      super( cb1in, cb2in );
    }

    /**
     * Evaluates the comparision and returns the result.
     */
    public boolean evaluate() throws Exception {
        Comparable n1 = (Comparable) cb1.execute();
        Comparable n2 = (Comparable) cb2.execute();
      return n1.lt(n2);
    }

}
//
//  ComparisonNE.java
//  whom
//
//  Created by Arvid Bessen on Fri Oct 03 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Implements the comparison for inequality.
 */
public class ComparisonNE extends Comparison {

    /**
     * Constructor, takes the two expressions to be compared as input.
```

```
     */
    public ComparisonNE( Expression cb1in, Expression cb2in ) {
      super( cb1in, cb2in );
    }

    /**
     * Evaluates the comparision and returns the result.
     */
    public boolean evaluate() throws Exception {
        Comparable n1 = (Comparable) cb1.execute();
        Comparable n2 = (Comparable) cb2.execute();
      return n1.ne(n2);
    }

}
//
//  Condition.java
//  whom
//
//  Created by Arvid Bessen on Fri Oct 03 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Abstract class for a condition.
 */
public abstract class Condition extends Expression {

    /**
     * Constructor
     */
    public Condition() {

    }

    /**
     * Abstract function that performs the evaluation of a condition.
     */
    public abstract boolean evaluate() throws Exception;

    /**
     * Function that performs the execution of a condition as
     * an expression.
     */
    public ObservableObject execute() throws Exception {
      Boolean boo = new Boolean(false);
      boo.setValue( evaluate() );
      return boo;
    }

    /**
     * Returns Boolean.
     */
    public int returntype() {
      return Type.Boolean;
    }

    /**
     * Returns true.
     */
    public boolean rvalue() {
      return true;
```

```
    }

    /**
     * Returns false.
     */
    public boolean lvalue() {
      return false;
    }

}
//
//  ConditionAnd.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * And condition depending on two expressions.
 */
public class ConditionAnd extends Condition {

    /**
     * The two expressions we will evaluate for this condition.
     */
    Expression o1, o2;

    /**
     * Constructor, taking the two expressions we will evaluate
     * for this condition as parameters.
     */
    public ConditionAnd( Expression o1in,
                  Expression o2in ) {
        o1 = o1in;
        o2 = o2in;
    }

    /**
     * Evaluates the condition.
     */
    public boolean evaluate() throws Exception {
        LogicValue lv1 = (LogicValue) o1.execute();
        LogicValue lv2 = (LogicValue) o2.execute();
      return lv1.and(lv2);
    }

    /**
     * Returns the dependencies of the two expressions.
     */
    public Collection getDependencies() throws Exception {
      LinkedList ll = new LinkedList();
      ll.addAll( o1.getDependencies() );
      ll.addAll( o2.getDependencies() );
      return ll;
    }
}
//
//  ConditionNot.java
//  whom
//
//  Created by Arvid Bessen on Fri Oct 03 2003.
```

```
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Not condition depending on the value of an expression.
 */
public class ConditionNot extends Condition {

    /**
     * The expression this condition depends on.
     */
    Expression expr;

    /**
     * Constructor, takes the expression this condition depends on
     * as parameter.
     */
    public ConditionNot( Expression exprin ) {
      expr = exprin;
    }

    /**
     * Evaluates the condition.
     */
    public boolean evaluate() throws Exception {
      ObservableObject oo = expr.execute();

      // Let's see if we can test it
      if( ! (oo instanceof LogicValue) )
          throw new Exception("I do not know how to test this object:
"+oo);
      return ((LogicValue) oo).not();
    }

    /**
     * Returns the dependencies of the expression.
     */
    public Collection getDependencies() throws Exception {
      LinkedList ll = new LinkedList();
      ll.addAll( expr.getDependencies() );
      return ll;
    }
}
//
//  ConditionOnExpression.java
//  whom
//
//  Created by Arvid Bessen on Fri Oct 03 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Condition depending on the value of an expression.
 */
public class ConditionOnExpression extends Condition {

    /**
     * The expression this condition depends on.
     */
```

```
    Expression expr;

    /**
     * Constructor, takes the expression this condition depends on
     * as parameter.
     */
    public ConditionOnExpression(Expression exprin) {
      expr = exprin;
    }

    /**
     * Evaluates the condition.
     */
    public boolean evaluate() throws Exception {
      ObservableObject oo = expr.execute();

      // Let's see if we can test it
      if( ! (oo instanceof LogicValue) )
          throw new Exception("I do not know how to test this object:
"+oo);
      return ((LogicValue) oo).test();
    }

    /**
     * Returns the dependencies of the expression.
     */
    public Collection getDependencies() throws Exception {
      //System.out.println("Trying to get the dependencies from:
"+expr);
      return expr.getDependencies();
    }
}
//
//  ConditionOr.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Or condition depending on two expressions.
 */
public class ConditionOr extends Condition {

    /**
     * The two expressions we will evaluate for this condition.
     */
    Expression o1, o2;

    /**
     * Constructor, taking the two expressions we will evaluate
     * for this condition as parameters.
     */
    public ConditionOr( Expression o1in,
                  Expression o2in ) {
        o1 = o1in;
        o2 = o2in;
    }

    /**
     * Evaluates the condition.
```

```
     */
    public boolean evaluate() throws Exception {
        LogicValue lv1 = (LogicValue) o1.execute();
        LogicValue lv2 = (LogicValue) o2.execute();
      return lv1.or(lv2);
    }

    /**
     * Returns the dependencies of the two expressions.
     */
    public Collection getDependencies() throws Exception {
      LinkedList ll = new LinkedList();
      ll.addAll( o1.getDependencies() );
      ll.addAll( o2.getDependencies() );
      return ll;
    }
}
//
//  ConstBooleanLookup.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * A constant boolean value.
 */
public class ConstBooleanLookup extends Lookup {
    /**
     * The value.
     */
    boolean b;

    /**
     * Constructor, takes the value as input.
     */
    public ConstBooleanLookup(boolean bin) {
      // this is not really clever, but...
      super( String.valueOf(bin) );
      b = bin;
    }

    /**
     * Returns either True or False.
     */
    public ObservableObject execute( ) {
      if( b )
          return new True();
      else
          return new False();
    }

    /**
     * None.
     */
    public Collection getDependencies() {
      return new LinkedList();
    }

    /**
     * Boolean.
```

```
     */
    public int returntype() {
      return Type.Boolean;
    }

    /**
     * True.
     */
    public boolean rvalue() {
      return true;
    }

    /**
     * False.
     */
    public boolean lvalue() {
      return false;
    }
}
//
//  ConstNumber.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * A constant number.
 */
public class ConstNumber extends Number {
    /**
     * Constructor, takes the value of this number as input.
     */
    public ConstNumber( int i ) {
        super(i, false);
    }

    /**
     * Constructor, takes the value of this number as input.
     */
    public ConstNumber( double d ) {
        super(d, false);
    }

    /**
     * This should not be called, since this is a constant string ;-)
     */
    public void setNumber( int i ) {
        // error, this is a constant
    }

    /**
     * This should not be called, since this is a constant string ;-)
     */
    public void setNumber( double d ) {
        // error, this is a constant
    }

    /**
     * This should not be called, since this is a constant string ;-)
     */
```

```
    ObservableObject assign( ObservableObject oo ) {
        // error, this is a constant
      return this;
    }

}
//
//  ConstNumberLookup.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * A constant number lookup.
 */
public class ConstNumberLookup extends Lookup {
    /**
     * The value that is to be looked up.
     */
    double d;

    /**
     * Constructor, takes the number that is supposed to be looked up as
input.
     */
    public ConstNumberLookup(double din) {
      // this is not really clever, but...
      super( String.valueOf(din) );
      d = din;
    }

    /**
     * Returns a ConstNumber with value d.
     */
    public ObservableObject execute( ) {
      return new ConstNumber(d);
    }

    /**
     * None.
     */
    public Collection getDependencies() {
      return new LinkedList();
    }

    /**
     * Number
     */
    public int returntype() {
      return Type.Number;
    }

    /**
     * True.
     */
    public boolean rvalue() {
      return true;
    }

    /**
```

```
     * False.
     */
    public boolean lvalue() {
      return false;
    }
}
//  ConstString.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * A constant string.
 */
public class ConstString extends WhomString {
    /**
     * Constructor, takes the value of this string as input.
     */
    public ConstString( String strin ) {
        super(strin, false);
    }

    /**
     * This should not be called, since this is a constant string ;-)
     */
    public void setString( String strin ) {
        // error, this is a constant
    }

    /**
     * This should not be called, since this is a constant string ;-)
     */
    ObservableObject assign( ObservableObject oo ) {
        // error, this is a constant
      return this;
    }

}
//
//  ConstStringLookup.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Returns a string constant.
 */
public class ConstStringLookup extends Lookup {
    /**
     * The string that should be returned.
     */
    String str;

    /**
     * Constructor, taking the string that should be returned as input.
```

```
     */
    public ConstStringLookup(String strin) {
      // this is not really clever, but...
      super( strin );
      str = strin;
    }

    /**
     * Returns a Whomstring object for str.
     */
    public ObservableObject execute( ) {
      return new ConstString(str);
    }

    /**
     * No dependencies, just constants.
     */
    public Collection getDependencies() {
      return new LinkedList();
    }

    /**
     * String.
     */
    public int returntype() {
      return Type.String;
    }

    /**
     * True.
     */
    public boolean rvalue() {
      return true;
    }

    /**
     * False.
     */
    public boolean lvalue() {
      return false;
    }
}
//
//  DependencyTracker.java
//  whom
//
//  Created by Arvid Bessen on Fri Oct 03 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Specifies the getDependencies call which should
 * return all ObservableObject this depends on.
 */
public interface DependencyTracker {

    /**
     * Returns a list of all objects this depends on.
     */
    public Collection getDependencies() throws Exception;
}
//
```

```
//  Dispatch.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * A dispatch statement.
 */
public class Dispatch extends Statement {
    /**
     * The expression that should be dispatched.
     */
    Expression expr;

    /**
     * Constructor, taking the expression that should be dispatched as
input.
     */
    public Dispatch(Expression exprin) {
      expr = exprin;
    }

    /**
     * Dispatches.
     */
    public ObservableObject execute( ) throws Exception {
      Boolean boo = (Boolean) expr.execute();
      boo.dispatch();
      return boo;
    }

}
//
//  Division.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Implements the division of two numbers
 */
public class Division extends ArithmeticOp {

    /**
     * Constructor
     */
    public Division(Expression s1in,
                Expression s2in) {
      super(s1in, s2in);
    }

    /**
     * Actually executes the division of
     * the expressions given in the constructor.
     */
```

```
    public ObservableObject execute( ) throws Exception {
      Number n1 = (Number) s1.execute();
      Number n2 = (Number) s2.execute();
      return n1.div(n2);
    }

}
//
//  EmptyScope.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * The first scope in the scope hierarchy.
 *
 * Aptly called EmptyScope, because it does not contain
 * any variables, objects, ... ;-)
 */
public class EmptyScope extends ParsingScope {

    /**
     * Constructor for an empty scope
     */
    public EmptyScope() {
      super( );
    }

    /**
     * Always returns false, since this does not contain any variables,
objects....
     */
    public boolean hasVariable( String name ) {
      return false;
    }

    /**
     * Always returns false, since this does not contain any variables,
objects....
     */
    public boolean hasObject( String name ) {
      return false;
    }

    /**
     * Returns an empty array.
     */
    public Collection lookupAllNames() {
      // empty list
      return new ArrayList();
    }

    /**
     * Returns an empty array.
     */
    public Collection lookupAllVarsNames() {
      // empty list
      return new ArrayList();
    }
```

```
    /**
     * Returns an empty array.
     */
    public Collection lookupAllObjsNames() {
      // empty list
      return new ArrayList();
    }

    /**
     * Returns an Lookup object for the variable
     * with name name.
     * Fails, because we do not have any variables.
     */
    public Lookup objLu(String name) {
      return null;
    }

    /**
     * Used to instantiate this, by creating an EmptyScopeInstantiated
     * and placing it on top of the InstantiatedScope hierarchy.
     */
    public void instantiate() {
      InstantiatedScope.setCurrent( new EmptyScopeInstantiated() );
    }
}
//
//  EmptyScopeInstantiated.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * The first scope in the scope hierarchy.
 *
 * Aptly called EmptyScopeInstantiated, because it does not
 * contain any variables, objects, ... ;-)
 */
public class EmptyScopeInstantiated extends InstantiatedScope {

    /**
     * Constructor for an empty scope
     */
    public EmptyScopeInstantiated() {
      super( );
    }

    /**
     * Always returns null, since this does not
     * contain any variables, objects....
     */
    public ObservableObject lookup(String name) {
      // fail
      return null;
    }

    /**
     * Returns an empty array.
     */
    public Collection lookupAll() {
      // empty list
```

```
      return new ArrayList();
    }

    /**
     * Returns an empty array.
     */
    public Collection lookupAllVars() {
      // empty list
      return new ArrayList();
    }

    /**
     * Returns an empty array.
     */
    public Collection lookupAllObjs() {
      // empty list
      return new ArrayList();
    }

}
//
//  Event.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * An event, a better name would probably be "Once", since this name is
 * used in Whom.
 *
 * If the values in our condition change and the condition holds,
 * the given code is executed.
 */
public class Event extends java.lang.Object implements Observer {
    /**
     * This represents the condition
     */
    Condition cond;

    /**
     * This is the code that should be executed
     */
    Statement body;

    /**
     * Construtor for an event, takes the condition to be tested
     * and the statement to be executed.
     *
     * Registers itself as an observer of all observable objects
     * that occur in condin.
     */
    public Event( Condition condin,
              Statement bodyin ) throws Exception {
      cond = condin;
      body = bodyin;
      Collection coll = cond.getDependencies();
      Iterator it = coll.iterator();
      // System.out.println(coll);
      while( it.hasNext() ) {
          ObservableObject oo = (ObservableObject) it.next();
```

```
            oo.addObserver(this);
        }
    }

    /**
     * Gets called, whenever an observable object, for which this
     * is registered, changes.
     *
     * Tests the condition with the new values and - depending on
     * the outcome - executes body.
     */
    public void update(java.util.Observable obj,
                       java.lang.Object changedAttrib) {
        try{
            //System.out.println("Testing the condition"+cond);
            if( cond.evaluate() ) {
              //System.out.println("Invoking block"+body);
              body.execute();
            }
        }
        catch( Exception e ) {
            whom.WHOMWalker.reportBKError(e, "could not evaluate condition
or execute block!");
        }
    }
}
//
//  Expression.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * The abstract base class for all expressions in whom.
 *
 * Main features: must have a method execute and should be able
 * to tell what their return type is, and whether it is an
 * rvalue and lvalue.
 */
public abstract class Expression extends java.lang.Object
    implements DependencyTracker {
    /**
     * Constructor, does nothing.
     */
    public Expression() {

    }

    /**
     * The method that is used to execute / evaluate an
     * expression. This must be implemented by the subclassed.
     */
    public abstract ObservableObject execute( ) throws Exception;

    /**
     * The return type (see class Type).
     */
    public abstract int returntype();

    /**
```

```
     * The return Attribute (more detailed, see class Attribute).
     *
     * This default implemenation has to be changed if necessary.
     */
    public Attribute returnAttribute() {
      return new Attribute( returntype() );
    }

    /**
     * Can this be used on the right side of an assignment?
     */
    public abstract boolean rvalue();

    /**
     * Can you assign to this expression?
     */
    public abstract boolean lvalue();
}
//
//  False.java
//  whom
//
//  Created by Arvid Bessen on Sat Oct 18 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * The Boolean value false.
 */
public class False extends Boolean {

    /**
     * Constructor.
     */
    public False() {
        super(false);
      value = false;
    }

    /**
     * Just ignored, you cannot set false.
     */
    public void setValue(boolean valuein) {
        // error, this is a constant
    }

    /**
     * Just ignored, you cannot assign to false.
     */
    ObservableObject assign( ObservableObject oo ) {
        // error, this is a constant
      return this;
    }
}
//
//  For.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;
```

```
import java.util.*;

/**
 * Class for a for statement.
 */
public class For extends whom.backend.Statement {

    /**
     * Excecuted before the loop is entered for the first time.
     */
    Statement cb_init;

    /**
     * Condition that determines if loop_body is executed.
     */
    Condition cond;

    /**
     * Executed after each execution of loop_body.
     */
    Statement cb_last_loop_code;

    /**
     * The statement that is executed as long as cond is true.
     */
    Statement loop_body;

    /**
     * Constructor, this class runs loop_bodyin as long as condin is
true.
     * cb_initin is excuted before the loop is entered,
cb_last_loop_codein
     * after each execution of loop_bodyin.
     */
    public For(Statement cb_initin,
             Condition condin,
             Statement cb_last_loop_codein,
             Statement loop_bodyin ) {
      cb_init = cb_initin;
      cond = condin;
      cb_last_loop_code =  cb_last_loop_codein;
      loop_body = loop_bodyin;
    }

    /**
     * Perform the for statement.
     */
    public ObservableObject execute( ) throws Exception {
      ObservableObject oo = null;
      for(
cb_init.execute();cond.evaluate();oo=cb_last_loop_code.execute() ) {
          loop_body.execute();
      }
      return oo;
    }

}
//
//  GlobalLookup.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
```

```
package whom.backend;

import java.util.*;

/**
 * Takes a name and returns a reference to an object in the global scope
when demanded.
 */
public class GlobalLookup extends Lookup {
    /**
     * A reference to the object we are supposed to look up.
     */
    ObservableObject oo;

    /**
     * Constructor for a look up object that looks up an object
     * of name namein in the scope gscoi.
     * Since this is a global scope, we can do a static look up right
now.
     */
    public GlobalLookup(String namein, GlobalScopeInstantiated gscoi) {
      super(namein);
      oo = gscoi.get(name);
      //System.out.println("Our variable for "+name+" is "+oo);
    }

    /**
     * Returns a reference to an ObservableObject.
     */
    public ObservableObject execute( ) {
      //System.out.println("Returning "+oo+" for "+name);
      return oo;
    }

    /**
     * The return type (see class Type).
     */
    public int returntype() {
      return oo.getType();
    }

    /**
     * The return Attribute (more detailed, see class Attribute).
     */
    public Attribute returnAttribute() {
      int returntype = returntype();
      if( returntype == Type.Object )
          return new Attribute(((Object) oo).getClassRef());
      else
          return new Attribute(returntype);
    }

    /**
     * True.
     */
    public boolean rvalue() {
      return true;
    }

    /**
     * True.
     */
    public boolean lvalue() {
      return true;
    }
```

```
}
//
//  GlobalScope.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * The global scope, contains all global variables.
 *
 * This scope contains the references to all global variables.
 * The variables are indexed by name.
 *
 * We have nested scopes. If a query is not succesful in this scope,
 * we query the surrounding block. Thus a variable in this scope can
 * shadow a variable in the surrounding scope with the same name.
 */
public class GlobalScope extends ParsingScope {
    /**
     * The reference to the instantiation.
     */
    GlobalScopeInstantiated gsi;

    /**
     * Constructor.
     */
    public GlobalScope() {
      super();
      gsi = new GlobalScopeInstantiated();
    }

    /**
     * Returns an Lookup object for the variable
     * with name name.
     */
    public Lookup objLu(String name) {
      return new GlobalLookup(name, gsi);
    }

    /**
     * Add a variable to this scope and immediately also
     * to the instantiated scope.
     */
    public void add(String name, Attribute attr) throws Exception {
      if( ! this.has(name) ) {
          // add one instantiation
          gsi.add(name, attr.create());

          // and some bookkeeping in case somebody asks for it
          vars.put(name, attr);
      }
      else
          throw new Exception("Variable with name "+name
                        +" already defined in this scope!");
    }

    /**
     * Makes this scope the current scope.
     *
     * May perform additional things to do this.
```

```
     * Should be overridden by subclasses that need this.
     */
    public void instantiate() {
      InstantiatedScope.setCurrent( gsi );
    }
}
//
//  GlobalScopeInstantiated.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * The global scope, contains all global variables.
 *
 * This scope contains the references to all global variables.
 * The variables are indexed by name.
 *
 * We have nested scopes. If a query is not succesful in this scope,
 * we query the surrounding block. Thus a variable in this scope can
 * shadow a variable in the surrounding scope with the same name.
 */
public class GlobalScopeInstantiated extends InstantiatedScope  {

    /**
     * Constructor, takes the surrounding scope as argument.
     */
    public GlobalScopeInstantiated() {
      super();
    }

    /**
     * Add a variable to this scope.
     */
    public void add(String name, ObservableObject obj) {
      vars.put(name, obj);
    }

}
//
//  IfThen.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Class for a if-then statement.
 */
public class IfThen extends whom.backend.Statement {

    /**
     * Condition that determines if cb is executed.
     */
    Condition cond;
```

```
    /**
     * The statement that is executed if cond is true.
     */
    Statement cb;

    /**
     * Constructor, this class executes cbin if condin is true.
     */
    public IfThen(Condition condin,
              Statement cbin) {
      cond = condin;
      cb = cbin;
    }

    /**
     * Perform the if-then statement.
     */
    public ObservableObject execute( ) throws Exception {
      if( cond.evaluate() )
          return cb.execute();
      else
          return null;
    }
}
//
//  IfThenElse.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Class for an if-then-else statement.
 */
public class IfThenElse extends whom.backend.Statement {

    /**
     * Condition that determines if cb1 or cb2 is executed.
     */
    Condition cond;

    /**
     * The statement that is executed if cond is true.
     */
    Statement cb1;

    /**
     * The statement that is executed if cond is not true.
     */
    Statement cb2;

    /**
     * Constructor, this class executed firstcb if condin is true
     * and secondcb if it is false.
     */
    public IfThenElse(Condition condin,
                  Statement firstcb,
                  Statement secondcb) {
      cond = condin;
      cb1 = firstcb;
      cb2 = secondcb;
```

```
    }

    /**
     * Perform the if-then-else statement.
     */
    public ObservableObject execute( ) throws Exception {
      if( cond.evaluate() )
          return cb1.execute();
      else
          return cb2.execute();
    }

}
//
//  InstantiatedScope.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * An InstantiatedScope is the mapping between names and actual objects.
 *
 * An InstantiatedScope is created by instiating a ParsingScope. This
 * will create all the variables that were defined in the ParsingScope.
 */
public class InstantiatedScope extends Scope {

    /**
     * A stack of scopes. The one on top gets actually used.
     */
    public static LinkedList scopes = new LinkedList();

    /**
     * Clear the collection of scopes.
     */
    public static void clearScopes ( ) {
      if ( scopes != null )
          scopes.clear();
    }

    static {
      scopes.add(new EmptyScopeInstantiated());
    }

    /**
     * The scope that is currently used.
     */
    public static InstantiatedScope current() {
      return (InstantiatedScope) scopes.getFirst();
    }

    /**
     * Adds a new InstantiatedScope to the stack of scopes and makes
     * it the current.
     */
    public static void setCurrent(InstantiatedScope isi) {
      scopes.addFirst(isi);
    }

    /**
```

```
     * Removes the current scope from the top of the stack and returns
it.
     */
    public static InstantiatedScope removeCurrent() {
      return (InstantiatedScope) scopes.removeFirst();
    }

    /**
     * Lookup in this scope, returns the object / variable for the name.
     */
    public ObservableObject lookup(String name) {
      //System.out.println("Trying to find "+name+" in "+this+" "+vars);
      return (ObservableObject) vars.get( name );
    }

    /**
     * Lookup a basic types in this scope with name name.
     */
    public ObservableObject lookupVariable(String name) {
      ObservableObject o = lookup(name);
      // System.out.println("Looking up "+name+" in "+this+" yields
"+o);
      if( isVariable(o) )
          return o;
      else
          return null;
    }

    /**
     * Lookup an object in this scope with name name.
     */
    public whom.backend.Object lookupObject(String name) {
      ObservableObject o = lookup(name);
      if( isObject(o) )
          return (whom.backend.Object) o;
      else
          return null;
    }

    /**
     * Returns all variables visible in this scope. Implementation.
     */
    protected Collection lookupAll() {
      return vars.values();
    }

    /**
     * Returns all variables of basic type (currently Number and String)
     * visible in this scope. Implementation.
     */
    protected Collection lookupAllVars() {
      LinkedList l = new LinkedList();
      Iterator it = lookupAll().iterator();
      while( it.hasNext() ) {
          ObservableObject o = (ObservableObject) it.next();
          if( isVariable(o) )
            l.add(o);
      }
      return l;
    }

    /**
     * Returns all variables of object type (this means instances of
classes
```

```
     * defined in a whom source file) visible in this scope.
Implementation.
     */
    protected Collection lookupAllObjs() {
      LinkedList l = new LinkedList();
      Iterator it = lookupAll().iterator();
      while( it.hasNext() ) {
          ObservableObject o = (ObservableObject) it.next();
          if( isObject(o) )
            l.add(o);
      }
      return l;
    }

    /**
     * Returns the variable in the current scope with this name.
     */
    static public ObservableObject get( String name ) {
      ObservableObject o;
      Iterator it = InstantiatedScope.scopes.iterator();
      while( it.hasNext() ) {
          InstantiatedScope isi = (InstantiatedScope) it.next();
          o = isi.lookup(name);
          if ( o != null )
            return o;
      }
      return null;
    }

    /**
     * Returns the variable of basic type (Number or String) in the
current
     * scopes with this name.
     */
    static public whom.backend.ObservableObject getVariable( String name
) {
      ObservableObject o;
      Iterator it = InstantiatedScope.scopes.iterator();
      while( it.hasNext() ) {
          InstantiatedScope isi = (InstantiatedScope) it.next();
          o = isi.lookup(name);
          // System.out.println("Looking up "+name+" in scope "+sco+" =
"+isi.vars+" yields "+o);
          if ( isi.isVariable(o) )
            return (whom.backend.ObservableObject) o;
      }
      return null;
    }

    /**
     * Returns the instance of a class in the current scope
     * with this name.
     */
    static public whom.backend.Object getObject( String name ) {
      ObservableObject o;
      Iterator it = InstantiatedScope.scopes.iterator();
      while( it.hasNext() ) {
          InstantiatedScope isi = (InstantiatedScope) it.next();
          o = isi.lookup(name);
          if ( isi.isObject(o) )
            return (whom.backend.Object) o;
      }
      return null;
    }
```

```
    /**
     * Returns all variables in the current scope.
     */
    static public Collection all() {
      LinkedList ll = new LinkedList();
      Iterator it = InstantiatedScope.scopes.iterator();
      while( it.hasNext() ) {
          InstantiatedScope isi = (InstantiatedScope) it.next();
          ll.addAll(isi.lookupAll());
      }
      return ll;
    }

    /**
     * Returns all variables of basic type (currently Number and String)
     * visible in  the current scope.
     */
    static public Collection allVariables() {
      LinkedList ll = new LinkedList();
      Iterator it = InstantiatedScope.scopes.iterator();
      while( it.hasNext() ) {
          InstantiatedScope isi = (InstantiatedScope) it.next();
          ll.addAll(isi.lookupAllVars());
      }
      return ll;
    }

    /**
     * Returns all variables of object type (this means instances of
classes
     * defined in a whom source file) visible in the current scope.
     */
    static public Collection allObjects() {
      LinkedList ll = new LinkedList();
      Iterator it = InstantiatedScope.scopes.iterator();
      while( it.hasNext() ) {
          InstantiatedScope isi = (InstantiatedScope) it.next();
          ll.addAll(isi.lookupAllObjs());
      }
      return ll;
    }

    /**
     * Returns all names of the variables visible in the current scope.
     */
    static public Collection allNames() {
      LinkedList ll = new LinkedList();
      Iterator it = InstantiatedScope.scopes.iterator();
      while( it.hasNext() ) {
          InstantiatedScope isi = (InstantiatedScope) it.next();
          ll.addAll(isi.lookupAllNames());
      }
      return ll;
    }

    /**
     * Returns all names of variables of basic type (currently Number
and
     * String) visible in the current scope.
     */
    static public Collection allVariablesNames() {
      LinkedList ll = new LinkedList();
      Iterator it = InstantiatedScope.scopes.iterator();
      while( it.hasNext() ) {
          InstantiatedScope isi = (InstantiatedScope) it.next();
```

```
            ll.addAll(isi.lookupAllVarsNames());
          }
          return ll;
        }

        /**
         * Returns all names of variables of object type (this means
instances of classes
         * defined in a whom source file) visible in the current scope.
         */
        static public Collection allObjectsNames() {
          LinkedList ll = new LinkedList();
          Iterator it = InstantiatedScope.scopes.iterator();
          while( it.hasNext() ) {
              InstantiatedScope isi = (InstantiatedScope) it.next();
              ll.addAll(isi.lookupAllObjsNames());
          }
          return ll;
        }

        /**
         * Returns true, if oo is not an object.
         */
        protected boolean isVariable(java.lang.Object oo) {
          return oo instanceof whom.backend.Number
              || oo instanceof whom.backend.Boolean
              || oo instanceof whom.backend.WhomString;
        }

        /**
         * Returns true, if oo is an object.
         */
        protected boolean isObject(java.lang.Object oo) {
          return oo instanceof whom.backend.Object;
        }

}

package whom;

import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;

/**
 * A sample interpreter.
 *
 * Should be working rudimentarily.
 */
public class Interpreter {

    public static void main(String[] args) {
      try {
            // Create lexer
          WHOMLexer lexer =
                new WHOMLexer(new DataInputStream(System.in));

          // Create parser
          WHOMParser parser = new WHOMParser(lexer);

          // Create walker
            WHOMWalker walker = new WHOMWalker();

            // Parse the input expression
            parser.program();
```

```
            CommonAST t = (CommonAST)parser.getAST();

            // Print the resulting tree out in LISP notation
            System.out.println(t.toStringList());

          // Get the backend ready
          whom.Backend.init();

            // Traverse the tree created by the parser
            walker.program(t);
        } catch(Exception e) {
            System.err.println("Exception\n"
                        +"Cause: "+e.getCause()+"\n"
                        +"Error message: "+e.getMessage()+"\n"
                        +"Stack trace:");
          e.printStackTrace();
        }

      System.out.println("Everything: "+
whom.backend.InstantiatedScope.current().allNames());
      System.out.println("All variables: "+
whom.backend.InstantiatedScope.current().allVariablesNames());
      System.out.println("All objects: "+
whom.backend.InstantiatedScope.current().allObjectsNames());

      if(
whom.backend.InstantiatedScope.current().allVariablesNames().contains("b
")
          &&
whom.backend.InstantiatedScope.current().allVariablesNames().contains("i
") ) {
          System.out.println("b before event: "+ ( (whom.backend.Number)
(whom.backend.InstantiatedScope.current().getVariable("b"))
).getNumber());

          System.out.println("Setting i to 5");
          ( (whom.backend.Number)
(whom.backend.InstantiatedScope.current().getVariable("i"))
).setNumber(5);

          whom.backend.ObservableObject.processEvents();

          System.out.println("b after: "+ ( (whom.backend.Number)
(whom.backend.InstantiatedScope.current().getVariable("b"))
).getNumber());
      }
    }
}
//
//  LogicValue.java
//  whom
//
//  Created by Arvid Bessen on Fri Oct 03 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Abstract interface for objects that behave in a boolean way.
 */
public interface LogicValue {

    /**
```

```
     * Is this object "true"?
     */
    public boolean test();

    /**
     * Are this object and the other object "true"?
     */
    public boolean and( LogicValue o );

    /**
     * Are this object or the other object "true"?
     */
    public boolean or( LogicValue o );

    /**
     * Is this object not "true"?
     */
    public boolean not();
}
//
//  Lookup.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Takes a name and returns a reference to an ObservableObject during
 * execution. This is abstract class for all other specializations
 * of a Lookup.
 */
public abstract class Lookup extends Expression {
    /**
     * The name we should look up.
     */
    String name;

    /**
     * Constructor, constructs a look up for the name passed.
     */
    public Lookup(String namein) {
      name = namein;
    }

    /**
     * Constructor, if no name is needed.
     */
    public Lookup() {
      name = null;
    }

    /**
     * Returns a reference to an ObservableObject.
     */
    public abstract ObservableObject execute( );

    /**
     * Returns the dependencies of the ObservableObject returned when
     * this is excecuted.
     */
    public Collection getDependencies() throws Exception {
```

```
      //System.out.println("Lookup "+this+" for "+name+" in scope
"+sco+" yields "+execute());
      //System.out.println("The dependencies:
"+execute().getDependencies());
      return ((DependencyTracker) execute()).getDependencies();
    }

}

package whom;

import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.debug.misc.ASTFrame;
//import src.*;

public class Main{

    public static void main(String [] args)
    {
      try{
          DataInputStream input = new DataInputStream(System.in);
          WHOMLexer lexer=new WHOMLexer(input);
          WHOMParser parser=new WHOMParser(lexer);
          parser.program();
          System.out.println("Start AST tree\n");
          CommonAST parseTree =(CommonAST)parser.getAST();
          System.out.println(parseTree.toStringList());
          ASTFrame frame = new ASTFrame("WHOM",parseTree);
          frame.setVisible(true);

          //lexer.nextToken();
      }
      catch(Exception e){}
    }

}
//
//  Method.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * A method as found in a class definition.
 */
public class Method extends java.lang.Object {

    /**
     * The name of the call
     */
    String methodname;

    /**
     * The return type (see Type for the meaning of different
constants).
     */
    int returnType;
```

```
    /**
     * The return class if the return type is an object.
     */
    Class returnTypeClass;

    /**
     * A list of all parameters in pairs, their name (String) and their
type (Attribute)
     */
    List parameters;

    /**
     * This is the Scope for the parameters
     */
    ParsingScope parameterScope;

    /**
     * This is the actual method body.
     */
    Block body = null;

    /**
     * The return value for all methods.
     */
    static ObservableObject returnvalue = null;

    /**
     * Constructor, constructs a method with the given return type
returnTypein
     * name name, a list of parameters as given by parametersin, and
     * the scope for the parameters parameterScopein.
     */
    public Method( int returnTypein,
                String name,
                List parametersin,
                ParsingScope parameterScopein ) throws Exception {
      methodname = name;
      returnType = returnTypein;
      returnTypeClass = null;
      parameters = parametersin;
      parameterScope = parameterScopein;
      body = null;
    }

    /**
     * Constructor, constructs a method with a return type object of the
     * passed returnTypeClassin,
     * name name, a list of parameters as given by parametersin and
     * the scope for the parameters parameterScopein.
     */
    public Method( Class returnTypeClassin,
                String name,
                List parametersin,
                ParsingScope parameterScopein ) throws Exception {
      methodname = name;
      returnType = Type.Object;
      returnTypeClass = returnTypeClassin;
      parameters = parametersin;
      parameterScope = parameterScopein;
      body = null;
    }

    /**
     * Sets the body of this method, the code that is actually executed
     * when the method is called.
```

```
     */
    public void setBody( Block bodyin ) {
      body = bodyin;
    }

    /**
     * The name of the method.
     */
    String getName() {
      return methodname;
    }

    /**
     * The number of parameters the method expects.
     */
    int getNumberOfParams() {
      return parameters.size() / 2;
    }

    /**
     * The return type.
     */
    int returnType() {
      return returnType;
    }

    /**
     * The return attribute.
     */
    Attribute returnAttribute() {
      if( returnType == Type.Object )
          return new Attribute(returnTypeClass);
      else
          return new Attribute(returnType);
    }

    /**
     * This method actually executes the method.
     */
    ObservableObject invoke( Object thisref, List parametersPassed )
throws Exception {
      // Here is the fun part...
      // System.out.println("Executing "+methodname);

      // Set all parameters

      // Test if the number of parameters is correct
      if( getNumberOfParams() != parametersPassed.size() )
          throw new Exception("Different number of parameters required
for method "+methodname);

      // Evaluate all parameters: applicative-order evaluation ;-)
      Iterator itPassed = parametersPassed.iterator();
      LinkedList llOoPassed = new LinkedList();
      while( itPassed.hasNext() ) {
          // get all passed parameters
          Expression expr = (Expression) itPassed.next();
          llOoPassed.addLast(expr.execute());
      }

      // Make parameters visible
      // System.out.println("Trying to instantiate the parameter scope:
"+parameterScope);
      parameterScope.instantiate();
```

```
        Iterator it = parameters.iterator();
        Iterator itOoPassed = llOoPassed.iterator();
        while( it.hasNext() && itOoPassed.hasNext() ) {
            // get all the variables where the parameters will be stored
            String paramName = (String) it.next();
            Attribute param = (Attribute) it.next();
            ObservableObject oo = InstantiatedScope.current().get(
paramName );

            ObservableObject ooPassed = (ObservableObject)
itOoPassed.next();

            // assign them to each other
            oo.assign(ooPassed);
        }

        // Create the returnvalue
        switch(returnType) {
        case Type.Number:
            returnvalue = new Number();
            break;
        case Type.String:
            returnvalue = new WhomString();
            break;
        case Type.Object:
            returnvalue = returnTypeClass.createObject();
            break;
        case Type.Void:
        default:
            returnvalue = null;
        }

        // execute the method's body
        try {
            body.execute();
        }
        catch( ReturnException e ) {
            // System.out.println("Return statement was executed");
            // Do nothing, we should just return
        }

        // Make parameters invisible again
        InstantiatedScope.removeCurrent();

        // return the value that is saved in the returnvalue
        return returnvalue;
    }

}
//
//  Methodcall.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * A class that represents a method call.
 */
public class Methodcall extends Expression {
    /**
```

```
     * The expression that should evaluate to the object the method
     * is called on.
     */
    Expression expr;

    /**
     * The actual name of the method.
     */
    String methodName;

    /**
     * A list of expressions that should evaluate to the parameters of
the
     * method.
     */
    List parameters;

    /**
     * Constructor for a method call.
     * Takes the following input:
     * the expression that should evaluate to the object the method
     * is called on,
     * the actual name of the method, and
     * a list of expressions that should evaluate to the parameters of
the
     * method.
     */
    public Methodcall(Expression exprin,
                  String methodNamein,
                  List parametersin) {
      expr = exprin;
      methodName = methodNamein;
      parameters = parametersin;
    }

    /**
     * Performs the method call.
     */
    public ObservableObject execute( ) throws Exception {
      Object obj = (whom.backend.Object) expr.execute();
      //System.out.println("Invoking method "+methodName+" on object
"+obj);
      return obj.invokeMethod(methodName, parameters);
    }

    /**
     * Returns the dependencies of the base object and all the
expressions
     * that are passed.
     */
    public Collection getDependencies() throws Exception {
      LinkedList ll = new LinkedList();
      ll.addAll( ((DependencyTracker) expr.execute()).getDependencies()
);
      Iterator it = parameters.iterator();
      while( it.hasNext() ) {
          ll.addAll( ((DependencyTracker) it.next()).getDependencies()
);
      }
      return ll;
    }

    /**
     * Returns the method's return type.
     */
```

```
    public int returntype() {
      return returnAttribute().getType();
    }

    /**
     * The return Attribute (more detailed, see class Attribute).
     */
    public Attribute returnAttribute() {
      Attribute att = expr.returnAttribute();
      if( att.getType() != Type.Object ) {
          // the method call will most likely fail
          return new Attribute(Type.Unknown);
      }
      return att.getClassref().methodReturnAttribute(methodName);
    }

    /**
     * True.
     */
    public boolean rvalue() {
      return true;
    }

    /**
     * False.
     */
    public boolean lvalue() {
      return false;
    }
}
//
//  Multiplication.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Implements the multiplication of two numbers
 */
public class Multiplication extends ArithmeticOp {

    /**
     * Constructor
     */
    public Multiplication(Expression s1in,
                          Expression s2in) {
      super(s1in, s2in);
    }

    /**
     * Actually executes the multiplication of
     * the expressions given in the constructor.
     */
    public ObservableObject execute( ) throws Exception {
      Number n1 = (Number) s1.execute();
      Number n2 = (Number) s2.execute();
      return n1.mul(n2);
    }

}
```

```
//
//  NoSuchVariableNameException.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

/**
 * Exception that is thrown when a variable cannot be found.
 */
public class NoSuchVariableNameException extends Exception {

    /**
     * Constructor, takes the name of the variable which could not
     * be found as parameter.
     */
    public NoSuchVariableNameException(String name) {
      super("Could not find variable with name: "+name+" .");
    }
}
//
//  Number.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * The numbers in the WHOM language.
 */
public class Number extends ObservableObject
    implements Comparable, LogicValue, DependencyTracker {

    /**
     * The actual value that is stored in this class.
     */
    double num = 0;

    /**
     * Constructor, constructs a non-observable zero.
     */
    public Number() {

    }

    /**
     * Constructor, constructs a non-observable number with value i.
     */
    public Number( int i ) {
        num = i;
    }

    /**
     * Constructor, constructs a non-observable number with value d.
     */
    public Number( double d ) {
        num = d;
    }
```

```
    /**
     * Constructor, constructs a observable (depending on observablein)
zero.
     */
    public Number(boolean observablein) {
      super(observablein);
    }

    /**
     * Constructor, constructs a observable (depending on observablein)
number with value i.
     */
    public Number( int i, boolean observablein) {
      super(observablein);
        num = i;
    }

    /**
     * Constructor, constructs a observable (depending on observablein)
number with value d.
     */
    public Number( double d, boolean observablein) {
      super(observablein);
        num = d;
    }

    /**
     * Number.
     */
    public int getType() {
      return Type.Number;
    }


    /**
     * Sets the value and notifies observers.
     */
    public void setNumber( int i ) {
      if( (double) i != num ) {
          num = (double) i;
          setChanged();
          notifyObserversLater(this);
      }
    }

    /**
     * Sets the value and notifies observers.
     */
    public void setNumber( double d ) {
      if( d != num ) {
          num = d;
          setChanged();
          notifyObserversLater(this);
      }
    }

    /**
     * The value that is stored in this object.
     */
    public double getNumber() {
      return num;
    }

    /**
     * Returns true if this is less than o
```

```
 */
public boolean lt( Comparable o ) {
  Number n = (Number) o;
    return num < n.num;
}

/**
 * Returns true if this is less or equal than o
 */
public boolean le( Comparable o ) {
  Number n = (Number) o;
    return num <= n.num;
}

/**
 * Returns true if this is equal to o
 */
public boolean eq( Comparable o ) {
  Number n = (Number) o;
    return num == n.num;
}

/**
 * Returns true if this is not equal to o
 */
public boolean ne( Comparable o ) {
  Number n = (Number) o;
    return num != n.num;
}

/**
 * Returns true if this is greater than o
 */
public boolean gt( Comparable o ) {
  Number n = (Number) o;
    return num > n.num;
}

/**
 * Returns true if this is greater or equal than o
 */
public boolean ge( Comparable o ) {
  Number n = (Number) o;
    return num >= n.num;
}

/**
 * True of this is greater than 0.
 */
public boolean test() {
  return num > 0;
}

/**
 * True of this is less or equal to 0.
 */
public boolean not() {
  return num <= 0;
}

/**
 * Are this object and the other object "true"?
 */
public boolean and(LogicValue lv) {
  return ((num>0) && lv.test());
```

```
    }

    /**
     * Are this object or the other object "true"?
     */
    public boolean or(LogicValue lv) {
      return ((num>0) || lv.test());
    }

    /**
     * Assign another number to this and notify observers.
     */
    ObservableObject assign( ObservableObject oo ) throws Exception {
      Number no = (Number) oo;
      setNumber( no.getNumber() );
      return this;
    }

    /**
     * Add the other number to this.
     */
    Number add( Number no ) {
      Number nr = new Number( num+no.num );
      return nr;
    }

    /**
     * Subtract the other number from this.
     */
    Number sub( Number no ) {
      Number nr = new Number( num-no.num );
      return nr;
    }

    /**
     * Multiply the other number and this.
     */
    Number mul( Number no ) {
      Number nr = new Number( num*no.num );
      return nr;
    }

    /**
     * Divide this by the other number.
     */
    Number div( Number no ) {
      Number nr = new Number( num/no.num );
      return nr;
    }

    /**
     * Returns a reference to this.
     */
    public Collection getDependencies() {
      LinkedList ll = new LinkedList();
      if( observable )
          ll.add(this);
      //System.out.println("The dependencies: "+ll);
      return ll;
    }
}
//
//  Object.java
//  whom
//
```

```
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * This class represents all objects for our WHOM language.
 *
 * It is an instantiation of a Class object and keeps all its variables
 * in an ObjectScope.
 */
public class Object extends ObservableObject
    implements DependencyTracker {

    /**
     * What class does this object belong to?
     */
    Class classref;

    /**
     * The scope for this object, here we have all variables for
     * this object.
     */
    ObjectScope objectScope;

    /**
     * What object are we currently in? This is set and used during
     * a method execution, see also ThisLookup, which uses exactly this
     * value.
     */
    static Object thisPointer;

    /**
     * Constructor for an object of the class that is given as
parameter.
     */
    public Object(Class classrefin) throws Exception {
      classref = classrefin;
      objectScope = new ObjectScope( classref.classScope );
    }

    /**
     * Clones this object by making a new copy and assigning
     * the values of all the attributes to the attributes of the new
copy.
     */
    public java.lang.Object clone() throws CloneNotSupportedException {
      try {
          Object nObj = (Object) super.clone();
          nObj.objectScope = new ObjectScope( classref.classScope );
          Iterator it = objectScope.allNames().iterator();
          while( it.hasNext() ) {
            String attributename = (String) it.next();
            nObj.objectScope.get(attributename).assign(objectScope.get(a
ttributename));
          }
          return nObj;
      }
      catch( CloneNotSupportedException cne ) {
          throw cne;
      }
      catch( Exception e ) {
          throw new CloneNotSupportedException();
```

```
  }
}

/**
 * Returns Object.
 */
public int getType() {
  return Type.Object;
}

/**
 * Sets the class reference.
 */
public void setClassRef( Class classrefin) {
  classref = classrefin;
}

/**
 * What class does this object belong to?
 */
public Class getClassRef() {
  return classref;
}

/**
 * Returns the attribute with the given name.
 */
public ObservableObject get( String name ) {
  return objectScope.lookup( name );
}

/**
 * Returns the attribute with the given name if it is a basic type.
 */
public ObservableObject getVariable( String name ) {
  return objectScope.lookupVariable( name );
}

/**
 * Returns the attribute with the given name if it is an object.
 */
public whom.backend.Object getObject( String name ) {
  return objectScope.lookupObject( name );
}

/**
 * Returns true if this object has an attribute of this name.
 */
public boolean has( String name ) {
  return get(name) != null;
}

/**
 * Returns true if this object has a basic attribute of this name.
 */
public boolean hasVariable( String name ) {
  return getVariable(name) != null;
}

/**
 * Returns true if this object has an object attribute of this name.
 */
public boolean hasObject( String name ) {
  return getObject(name) != null;
}
```

```
    /**
     * All attributes of this object.
     */
    public Collection all() {
      return objectScope.all();
    }

    /**
     * All basic attributes of this object.
     */
    public Collection allVariables() {
      return objectScope.allVariables();
    }

    /**
     * All attributes which are objects themselves of this object.
     */
    public Collection allObjects() {
      return objectScope.allObjects();
    }

    /**
     * Invokes one of the methods of this object with the given
parameter.
     */
    public ObservableObject invokeMethod( String name, List parameters )
throws Exception {
      // Let's put our variables in place...
      // Save the old this pointer
      Object oldthisPointer = thisPointer;

      // Now we are the current object
      thisPointer = this;

      // .. call the method...
      ObservableObject oo = classref.invoke( this, name, parameters );

      // ... and take our variables out of sight again
      thisPointer = oldthisPointer;

      return oo;
    }

    /**
     * Assigns the passed object to this if the passed object is an
     * instantiation of the same class as this.
     */
    ObservableObject assign( ObservableObject oo ) throws Exception {
      // Let's see if this is an object as well...
      Object o = (Object) oo;

      // Which class does o belong to?
      Class ocr = o.classref;
      while( ocr != classref && ocr != null )
          ocr = ocr.superclass;

      if( ocr == null ) {
          // Bad luck...
          // throw some exception
          throw new Exception("Could not assign an object of class
"+o.classref.getClassname()+" to an object of class
"+classref.getClassname());
      }
      else {
```

```
          Collection coll = objectScope.allNames();
          Iterator it = coll.iterator();
          while( it.hasNext() ) {
            String attribname = (String) it.next();
            get( attribname ).assign( o.get( attribname ) );
          }
          return this;
      }
    }

    /**
     * Returns the dependencies of all attributes.
     */
    public Collection getDependencies() throws Exception {
      LinkedList ll = new LinkedList();
      // Depends on all attributes
      Collection attribs = all();
      Iterator it = attribs.iterator();
      while( it.hasNext() ) {
          ll.addAll( ( (DependencyTracker) it.next() ).getDependencies()
);
      }
      return ll;
    }

}
//
//  ObjectMemberLookup.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Used to get the members of an object.
 *
 * Takes a lookup for the object and a list of members of that
 * object to descend.
 */
public class ObjectMemberLookup extends Lookup {
    /**
     * The lookup that yields the starting object.
     */
    Lookup lu;

    /**
     * The list of all the member's names we should descend along.
     */
    Collection membernames;

    /**
     * Constructor, takes
     * the lookup that yields the starting object, and
     * the list of all the member's names we should descend along
     * as parameters.
     */
    public ObjectMemberLookup(Lookup luin, Collection membernamesin) {
      lu = luin;
      membernames = membernamesin;
    }
```

```
    /**
     * Returns the ObservableObject we were looking for.
     */
    public ObservableObject execute( ) {
      // System.out.println("Trying to get the first object");
      ObservableObject ret = lu.execute();
      // System.out.println("Found "+ret);
      Iterator it = membernames.iterator();
      while( it.hasNext() ) {
          String name = (String) it.next();
          // descend one level, i.e lookup y in the current object
          // System.out.println("Trying to descend: "+name);
          ret = ((whom.backend.Object) ret).get(name);
          // System.out.println("Found "+ret);
      }
      return ret;
    }

    /**
     * Returns the dependencies of the the object we look up.
     */
    public Collection getDependencies() throws Exception {
      return ((DependencyTracker) execute()).getDependencies();
    }

    /**
     * The type of the object that will be returned.
     */
    public int returntype() {
      return returnAttribute().getType();
    }

    /**
     * The return Attribute (more detailed, see class Attribute).
     */
    public Attribute returnAttribute() {
      Attribute attr = lu.returnAttribute();
      Iterator it = membernames.iterator();
      while( it.hasNext() ) {
          String name = (String) it.next();
          if( attr.getType() != Type.Object )
            return new Attribute(Type.Unknown);
          attr = attr.getClassref().attributeType( name );
      }
      return attr;
    }

    /**
     * True.
     */
    public boolean rvalue() {
      return true;
    }

    /**
     * True.
     */
    public boolean lvalue() {
      return true;
    }
}
//  ObjectScope.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
```

```
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * This is the scope for an object. Everything that is in this scope
 * will have to be referenced indirectly via a "this" pointer
 *
 * The ObjectScope class is the instantiation of a ClassScope.
 */
public class ObjectScope extends InstantiatedScope {

    /**
     * Constructor, constructs the objectscope for
     * an object of the class with the ClassScope passed as parameter.
     */
    public ObjectScope(ClassScope classScope) throws Exception {
      // System.out.println("Creating ObjectScope for ClassScope "
      // +classScope+" "+classScope.vars+" from class
"+classScope.classref.classname);
      classScope.classref.addAttributesToObjectScope(this);
    }

    /**
     * Add a variable to this scope.
     */
    public void add(String name, Attribute attr) throws Exception {
      add(name, attr.create());
    }

    /**
     * Add a variable to this scope.
     */
    public void add(String name, ObservableObject obj) {
      vars.put(name, obj);
    }

}
//
//  ObservableObject.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * The basic abstract class for all variables and objects in whom.
 */
public abstract class ObservableObject
    extends java.util.Observable
    implements Cloneable {

    /**
     * this is the event queue
     */
    static LinkedList eventqueue = new LinkedList();

    /**
     * do we actually want to be observed??
```

```
     */
    boolean observable;

    /**
     * do we have an event pending that will be executed?
     */
    boolean eventPending = false;

    /**
     * Constructor, creates an observable object, observable by default.
     */
    public ObservableObject() {
      observable = true;
    }

    /**
     * Constructor, creates an observable object, observablein
determines
     * whether it is observable.
     */
    public ObservableObject(boolean observablein) {
      observable = observablein;
    }

    /**
     * Creates a clone of this.
     */
    public java.lang.Object clone() throws CloneNotSupportedException {
      // shallow copy should be enough
      // System.out.println("Cloning "+this);
      java.lang.Object nObj =  super.clone();
      return nObj;
    }

    /**
     * Returns whether or not this object is observable.
     */
    public boolean isObservable() {
      return observable;
    }

    /**
     * Returns the type of this object (see class Type).
     */
    abstract public int getType();


    /**
     * Assigns another ObservableObject to this. Abstract, has to
implemented
     * by concrete subclasses.
     */
    abstract ObservableObject assign( ObservableObject oo ) throws
Exception;

    /**
     * The basic mechanism for processing events: whenever an
ObservableObject
     * changes it calls this method.
     *
     * This builds upon the notifyObservers / update mechanism of java's
     * Observable and Observer classes, but
     * notifyObservers is immediate, so we changed that
     * to a queue
     */
```

```
    protected void notifyObserversLater( java.lang.Object o ) {
      // I am lazy so let us just put our two parameters one
      // after the other into the queue
      //    if( observable )
      //        System.out.println("We notify later for "+this
      //                        +", which "+(eventPending?"has":"has
not")
      //                        +" an event pending");
      if( observable && ! eventPending ) {
          eventqueue.addLast( this );
          eventqueue.addLast( o );
          eventPending = true;
      }
    }

    /**
     * process all events in the queue.
     */
    public static void processEvents() {
      while( ! eventqueue.isEmpty() ) {
          ObservableObject oo = (ObservableObject)
eventqueue.removeFirst();
          java.lang.Object o = eventqueue.removeFirst();
          oo.eventPending = false;
          oo.notifyObservers( o );
      }
    }

}
//
//  ParsingScope.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * One of the two specializations of a scope, ParsingScope is used
during
 * parsing the program to keep track of all the names and the associated
 * variables and objects that will have to
 * be created when the block is actually entered during execution.
 */
abstract public class ParsingScope extends Scope {

    /**
     * A stack of scopes. The one on top gets actually used.
     */
    public static LinkedList scopes = new LinkedList();

    /**
     * Clear the collection of scopes.
     */
    public static void clearScopes ( ) {
      if ( scopes != null )
          scopes.clear();
    }

    static {
      scopes.add(new EmptyScope());
    }
```

```
    /**
     * The scope that is currently used.
     */
    public static ParsingScope current() {
      return (ParsingScope) scopes.getFirst();
    }

    /**
     * Adds a new ParsingScope to the stack of scopes and makes
     * it the current.
     */
    public static void setCurrent(ParsingScope psi) {
      scopes.addFirst(psi);
    }

    /**
     * Removes the current scope from the top of the stack and returns
it.
     */
    public static ParsingScope removeCurrent() {
      return (ParsingScope) scopes.removeFirst();
    }

    /**
     * Returns an Lookup object for the variable
     * with name name.
     */
    public Lookup lu(String name) throws NoSuchVariableNameException {
      ObservableObject o = null;
      Iterator it = ParsingScope.scopes.iterator();
      while( it.hasNext() ) {
          ParsingScope psi = (ParsingScope) it.next();
          if ( psi.has(name) )
            return psi.objLu(name);
      }
      throw new NoSuchVariableNameException(name);
    }

    /**
     * Constructs the approriate Lookup
     */
    protected abstract Lookup objLu(String name);

    /**
     * Add a variable to this scope.
     */
    public void add(String name, Attribute attr) throws Exception {
      if( ! this.has(name) )
          vars.put(name, attr);
      else
          throw new Exception("Variable with name "+name
                        +" already defined in this scope!");
    }

    /**
     * Returns the Attribute of the variable with this name in this
scope.
     */
    public Attribute lookupAttribute(String name) {
      return (Attribute) vars.get(name);
    }

    /**
```

```
     * Returns the attribute of the variable with this name in the
current scope.
     */
    static public Attribute getAttribute(String name) {
      Iterator it = ParsingScope.scopes.iterator();
      while( it.hasNext() ) {
          ParsingScope psi = (ParsingScope) it.next();
          if( psi.has(name) )
            return psi.lookupAttribute(name);
      }
      return new Attribute(Type.Unknown);

    }

    /**
     * Returns all names of the variables visible in the current scope.
     */
    static public Collection allNames() {
      LinkedList ll = new LinkedList();
      Iterator it = ParsingScope.scopes.iterator();
      while( it.hasNext() ) {
          ParsingScope psi = (ParsingScope) it.next();
          ll.addAll(psi.lookupAllNames());
      }
      return ll;
    }

    /**
     * Returns all names of variables of basic type (currently Number
and
     * String) visible in the current scope.
     */
    static public Collection allVariablesNames() {
      LinkedList ll = new LinkedList();
      Iterator it = ParsingScope.scopes.iterator();
      while( it.hasNext() ) {
          Scope sco = (Scope) it.next();
          ll.addAll(sco.lookupAllVarsNames());
      }
      return ll;
    }

    /**
     * Returns all names of variables of object type (this means
instances
     * of classes
     * defined in a whom source file) visible in the current scope.
     */
    static public Collection allObjectsNames() {
      LinkedList ll = new LinkedList();
      Iterator it = ParsingScope.scopes.iterator();
      while( it.hasNext() ) {
          Scope sco = (Scope) it.next();
          ll.addAll(sco.lookupAllObjsNames());
      }
      return ll;
    }

    /**
     * Returns true, if oo is not an object.
     */
    protected boolean isVariable(java.lang.Object oo) {
      return ((Attribute) oo).isVariable();
    }
```

```
    /**
     * Returns true, if oo is an object.
     */
    protected boolean isObject(java.lang.Object oo) {
      return ((Attribute) oo).isObject();
    }

    /**
     * Makes this scope the current instantiated scope.
     *
     * May perform additional things to do this.
     * Should be overridden by subclasses that need this.
     *
     * See the associated class InstantiatedScope for more information.
     */
    public abstract void instantiate() throws Exception;
}
//
//  Return.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * A return statement to abort execution of the current method
 * and return a value.
 */
public class Return extends Statement {

    /**
     * The value that should be returned will be generated from
     * evaluating this expression.
     */
    Expression retval;

    /**
     * Constructor, takes the expression to be evaluated and to
     * return as parameter.
     */
    public Return(Expression retvalin) {
      retval = retvalin;
    }

    /**
     * Executed the actual return by throwing an ReturnException after
     * evaluating retval and assigning it to Method.returnvalue.
     */
    public ObservableObject execute( ) throws Exception {
      ObservableObject retvalobj = retval.execute();
      //System.out.println("Returning "+retvalobj);
      Method.returnvalue.assign( retvalobj );
      throw new ReturnException();
    }

}
//
//  ReturnException.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
```

```
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

/**
 * Returnexception is used to deviate from the normal execution
 * path and thrown by a return statement.
 * It will be caught by the method that was invoked.
 */
public class ReturnException extends Exception {

    /**
     * Constructor.
     */
    public ReturnException() {
      super();
    }
}
//
//  RootClass.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;


/**
 * A special class that is the root of whom's class hierarchy.
 *
 * This class is used as a "terminator" for lookups that search for
 * methods.
 */
public class RootClass extends Class {

    /**
     * Constructor.
     */
    public RootClass() {
      super("RootClass", null, new ClassScope());
    }

    /**
     * This is impossible, it should never be called.
     */
    public void addAttributesToObjectScope(ObjectScope os) {
      // do nothing
    }

    /**
     * Does nothing, no variables for this class.
     */
    protected void invokeConstructor() {
      // do nothing
    }

    /**
     * The return type of the method with this name.
     */
    public int methodReturnType(String methodname) {
      return Type.Unknown;
    }
```

```
    /**
     * The return attribute of the method with this name.
     */
    public Attribute methodReturnAttribute(String methodname) {
      return new Attribute(Type.Unknown);
    }

    /**
     * This class does not have any methods.
     */
    public Method getMethod( String methodname,
                         int numberOfParameters ) throws Exception {
      // fail
      throw new Exception("Could not find method \""+methodname+
                      "\" which takes "+numberOfParameters+
                      " parameter(s)");
    }

    /**
     * This class does not have any methods.
     */
    public ObservableObject invoke( Object thisref,
                            String methodname,
                            List parameters ) throws Exception {
      // fail
      throw new Exception("Could not find method \""+methodname+
                      "\" which takes "+parameters.size()+
                      " parameter(s)");
    }
}
//
//  Scope.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * The abstract scope superclass.
 *
 * A scope contains the references to all variables in a block.
 * The variables are indexed by name.
 *
 * We have nested scopes. If a query is not succesful in this scope,
 * we query the enclosing block. Thus a variable in this scope can
 * shadow a variable in the enclosing scope with the same name.
 *
 * This is the abstract superclass for specializations that implement
 * different behavior.
 */
abstract public class Scope extends java.lang.Object {

    /**
     * All our mappings of names to objects / variables are stored here.
     */
    HashMap vars;

    /**
     * Constructor, creates an initally empty scope.
     */
```

```
    public Scope() {
      vars = new HashMap();
    }

    /**
     * Returns true, if the scope contains
     * something of this name.
     */
    public boolean has( String name ) {
      java.lang.Object o = vars.get(name);
      return o != null;
    }

    /**
     * Returns true, if the scope contain
     * a basic type (Number or String) with this name.
     */
    public boolean hasVariable( String name ) {
      java.lang.Object o = vars.get(name);
      return o != null && isVariable(o);
    }

    /**
     * Returns true, if the scope contains
     * am object of this name.
     */
    public boolean hasObject( String name ) {
      java.lang.Object o = vars.get(name);
      return o != null && isObject(o);
    }

    /**
     * Returns all names of the variables visible in this scope.
Implementation
     */
    public Collection lookupAllNames() {
      return vars.keySet();
    }

    /**
     * Returns all names of variables of basic type (currently Number
and
     * String) visible in this scope. Implementation.
     */
    public Collection lookupAllVarsNames() {
      LinkedList l = new LinkedList();
      Iterator it = lookupAllNames().iterator();
      while( it.hasNext() ) {
          String name = (String)it.next();
          if( isVariable(vars.get(name)) )
            l.add(name);
      }
      return l;
    }

    /**
     * Returns all names of variables of object type
     * (this means instances of classes
     * defined in a whom source file) visible in this scope.
Implementation.
     */
    public Collection lookupAllObjsNames() {
      // System.out.println("Looking for objects in "+this+" "+vars);
      LinkedList l = new LinkedList();
      Iterator it = lookupAllNames().iterator();
```

```
      while( it.hasNext() ) {
          String name = (String)it.next();
          if( isObject(vars.get(name)) )
            l.add(name);
      }
      return l;
    }

    /**
     * Returns true, if oo is not an object but a basic type.
     * Must be implemented by the subclass.
     */
    abstract protected boolean isVariable(java.lang.Object oo);

    /**
     * Returns true, if oo is an object.
     */
    abstract protected boolean isObject(java.lang.Object oo);
}
//
//  Statement.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * A statement. This class is the abstract superclass for all
 * statements, including the more complicated as blocks, while
 * loops, etc.
 */
public abstract class Statement extends java.lang.Object {

    /**
     * Constructor.
     */
    public Statement() {

    }

    /**
     * The abstract method that implements the execution of
     * a statement.
     */
    public abstract ObservableObject execute( ) throws Exception;

}
//
//  Subtraction.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Implements the subtraction of two numbers
 */
```

```
public class Subtraction extends ArithmeticOp {

    /**
     * Constructor
     */
    public Subtraction(Expression s1in,
                   Expression s2in) {
      super(s1in, s2in);
    }

    /**
     * Actually executes the subtraction of
     * the expressions given in the constructor.
     */
    public ObservableObject execute( ) throws Exception {
      Number n1 = (Number) s1.execute();
      Number n2 = (Number) s2.execute();
      return n1.sub(n2);
    }

}
//
//  ThisLookup.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * ThisLookup is used to get the indirection of a this
 * reference in methods.
 */
public class ThisLookup extends Lookup {

    /**
     * The class this will return.
     */
    Class classref;

    /**
     * Constructor - takes the Class as argument that will
     * be returned by this lookup.
     */
    public ThisLookup(Class classrefin) {
      super("");
      classref = classrefin;
      //System.out.println("ThisLookup created for"+name);
    }

    /**
     * Returns a reference to the object whose method is currently
called.
     */
    public ObservableObject execute( ) {
      // Get this pointer - that's all!
      return Object.thisPointer;
    }

    /**
     * Returns Type.Object
     */
```

```
    public int returntype() {
      return Type.Object;
    }

    /**
     * Returns an Attribute of Type.Object and with Class classref
     */
    public Attribute returnAttribute() {
      return new Attribute(classref);
    }

    /**
     * true
     */
    public boolean rvalue() {
      return true;
    }

    /**
     * true
     */
    public boolean lvalue() {
      return true;
    }
}
//
//  True.java
//  whom
//
//  Created by Arvid Bessen on Sat Oct 18 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * The Boolean value true.
 */
public class True extends Boolean {

    /**
     * Constructor.
     */
    public True() {
        super(false);
      value = true;
    }

    /**
     * Just ignored, you cannot set true.
     */
    public void setValue(boolean valuein) {
        // error, this is a constant
    }

    /**
     * Just ignored, you cannot assign to true.
     */
    ObservableObject assign( ObservableObject oo ) {
        // error, this is a constant
      return this;
    }
}
//
```

```
//  Type.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

/**
 * Just an awkward way to simulate #define
 *
 * Here every supported type is given a numeric value.
 *
 */
public class Type extends java.lang.Object {

    public static final int Unknown = 0;
    public static final int Number = 1;
    public static final int String = 2;
    public static final int Object = 3;
    public static final int Void = 4;
    public static final int Boolean = 5;

}
//
//  While.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Class for a while loop.
 */
public class While extends whom.backend.Statement {
    /**
     * Condition that determines if stat is executed.
     */
    Condition cond;

    /**
     * The statement that is executed as long as cond is true.
     */
    Statement stat;

    /**
     * Constructor, this class runs statin as long as condin is true.
     */
    public While(Condition condin,
                 Statement statin) {
      cond = condin;
      stat = statin;
    }

    /**
     * Perform the while operation.
     */
    public ObservableObject execute( ) throws Exception {
      ObservableObject oo = null;
      while( cond.evaluate() )
          oo = stat.execute();
```

```
      return oo;
    }
}
/**
 * Exception class: messages are generated in frontend and backend
   Three levels of errors are defined: warning, fatal error and runtime
error
 * Author: Rui Kuang
 * Date: 11/13/2003
 * Revise history:

**/
package whom;
import java.util.Vector;
import antlr.RecognitionException;
import java.io.*;

public class WHOMException extends RuntimeException {
    Vector errmsgs = new Vector();
    Vector warnings= new Vector();
    Vector runtime = new Vector();
    void adderr( String msg ) {
        whom.Backend.log(msg);
        errmsgs.add(msg);
    }
    void adderr( RecognitionException e ) {
        whom.Backend.log(e.toString());
        errmsgs.add(e.toString());
    }
    void addwarn( String msg ) {
        whom.Backend.log(msg);
        warnings.add(msg);
    }
    void addwarn( RecognitionException e ) {
        whom.Backend.log(e.toString());
        warnings.add(e.toString());
    }
    void addrun( String msg ) {
        whom.Backend.log(msg);
        runtime.add(msg);
    }
    void addrun( RecognitionException e ) {
        whom.Backend.log(e.toString());
        runtime.add(e.toString());
    }
}
//
//  WhomString.java
//  whom
//
//  Created by Arvid Bessen on Sun Oct 05 2003.
//  Copyright (c) 2003 __MyCompanyName__. All rights reserved.
//
package whom.backend;

import java.util.*;

/**
 * Class representing strings in WHOM
 */
public class WhomString extends ObservableObject
    implements Comparable, LogicValue, DependencyTracker {
    /**
     * Internal representation of the string.
     */
```

```
    String str = "";

    /**
     * Constructor, constructs the empty string, nonobservable.
     */
    public WhomString() {

    }

    /**
     * Constructor, constructs a WhomString representing
     * the string passed, nonobservable.
     */
    public WhomString( String strin ) {
        str = strin;
    }

    /**
     * Constructor, constructs the empty string, observability depends
     * on the parameter passed.
     */
    public WhomString(boolean observablein) {
      super(observablein);
    }

    /**
     * Constructor, constructs a WhomString representing
     * the string passed, observability depends
     * on the parameter passed.
     */
    public WhomString( String strin, boolean observablein) {
      super(observablein);
        str = strin;
    }

    /**
     * Creates a copy of this with a copy of the string str
     */
    public java.lang.Object clone() throws CloneNotSupportedException {
      WhomString nObj = (WhomString) super.clone();
      nObj.str = new String(str);
      return nObj;
    }

    /**
     * Returns the number representing a WhomString.
     */
    public int getType() {
      return Type.String;
    }

    /**
     * Sets this WhomString to another string and informs observers of
that.
     */
    public void setString( String strin ) {
      if( ! str.equals(strin) ) {
          str = new String(strin);
          setChanged();
          notifyObserversLater(this);
      }
    }

    /**
     * Returns the string that is represented by this object.
```

```
     */
    public String getString() {
      return str;
    }

    /**
     * Compare the two strings, returns true if this is less than the
     * WhomString passed.
     */
    public boolean lt( Comparable o ) {
      WhomString str2 = (WhomString) o;
        return str.compareTo(str2) < 0;
    }

    /**
     * Compare the two strings, returns true if this is less or equal
the
     * WhomString passed.
     */
    public boolean le( Comparable o ) {
      WhomString str2 = (WhomString) o;
        return str.compareTo(str2) <= 0;
    }

    /**
     * Compare the two strings, returns true if this is equal to
     * WhomString passed.
     */
    public boolean eq( Comparable o ) {
      WhomString str2 = (WhomString) o;
        return str.compareTo(str2) == 0;
    }

    /**
     * Compare the two strings, returns true if this is not equal to
     * WhomString passed.
     */
    public boolean ne( Comparable o ) {
      WhomString str2 = (WhomString) o;
        return str.compareTo(str2) != 0;
    }

    /**
     * Compare the two strings, returns true if this is greater than the
     * WhomString passed.
     */
    public boolean gt( Comparable o ) {
      WhomString str2 = (WhomString) o;
        return str.compareTo(str2) > 0;
    }

    /**
     * Compare the two strings, returns true if this is greater or equal
     * than the WhomString passed.
     */
    public boolean ge( Comparable o ) {
      WhomString str2 = (WhomString) o;
        return str.compareTo(str2) >= 0;
    }

    /**
     * Returns true, if this is not the empty string.
     */
    public boolean test() {
      return str.length() != 0;
```

```
    }

    /**
     * Returns true, if this is the empty string.
     */
    public boolean not() {
        return ! test();
    }

    /**
     * Returns true, if this is not the empty string and the parameter
     * passed evaluates to true.
     */
    public boolean and(LogicValue lv) {
      return ( test() && lv.test() );
    }

    /**
     * Returns true, if this is not the empty string or the parameter
     * passed evaluates to true.
     */
    public boolean or(LogicValue lv) {
      return ( test() || lv.test() );
    }

    /**
     * Assigns the WhomString oo to this and informs observers.
     */
    ObservableObject assign( ObservableObject oo ) throws Exception {
      WhomString strin = (WhomString) oo;
      setString( strin.getString() );
      return this;
    }

    /**
     * Returns a list containing this, if this is observable,
     * an empty list otherwise.
     */
    public Collection getDependencies() {
      LinkedList ll = new LinkedList();
      if( observable )
          ll.add(this);
      //System.out.println("The dependencies: "+ll);
      return ll;
    }
}
```

WEM files:

```
/*************************************************************************
******
 *
 *  WEM - [W]hom [EM]ulator
 *  version 0.01
 *  Fall 2003
 *
 *  Andrey Butov
 *
 *************************************************************************
******/

package wem;

import javax.swing.*;
import java.awt.*;
import java.util.*;
import java.io.*;
```

```
import java.awt.event.*;

public class WEM extends WindowAdapter {

    /** Application entry point. */
    public static void main ( String[] args ) {
            if ( args.length > 0 )
            application = new WEM ( args[0] );
            else
                  application = new WEM ( null );
    }

    /** WEM constructor. */
    public WEM ( String fileToOpen ) {
            initialize ( fileToOpen );
    }

      /** Send a log message to the log screen. */
      void log ( String s ) {
            if ( mainWindow != null )
                  mainWindow.log ( s );
      }

    /** Returns the main windows of the application. */
    WEMMainWindow getMainWindow ( ) {
        return mainWindow;
    }

      /** Returns the main wem/whom interface. */
      WEMWhomInterface getInterface() {
            return whomInterface;
      }

    /** Returns the currently loaded WHOM source file. */
    File getCurrentlyLoadedSourceFile() {
        return currentWhomFile;
    }

      /** Returns the directory of the currently loaded WHOM source
file. */
      File getCurrentlyLoadedSourceFileDirectory() {
            return currentWhomFileDirectory;
      }

    /** Initialize the WEM application. */
    private void initialize ( String fileToOpen ) {
            whomInterface = new WEMWhomInterface(this);
        mainWindow = new WEMMainWindow ( this );
        mainWindow.addWindowListener ( this );
        mainWindow.setVisible(true);

            try {
                  if ( fileToOpen != null ) {
                        File f = new File(fileToOpen);
                        newSourceFileChosen ( f );
                  }
            }
            catch ( NullPointerException e ) {
                  currentWhomFile = null;
            }
    }

    /** Respond to the main window closing event. */
    public void windowClosing(WindowEvent e) {
```

```
        // TODO: Making the assumption that only the main window is
sending this.
        getMainWindow().dispose();
        System.exit(0);
    }

      /** Reset all components of WEM. */
      private void resetAll ( ) {
            mainWindow.reset ( );
      }

    /** The user has chosen a new WHOM source file. */
    void newSourceFileChosen ( File chosenFile ) {
        resetAll ( );
            currentWhomFile = chosenFile;
      try {
                  whomInterface.start(chosenFile);
            }
            catch ( Exception e ) {
                  mainWindow.whomError(e);
            }
      }

    private WEMMainWindow     mainWindow;
      private WEMWhomInterface      whomInterface;
    private File              currentWhomFile;
      private File                        currentWhomFileDirectory;
    private static WEM        application;
}
/**************************************************************************
*******
 *
 * WEM - [W]hom [EM]ulator
 * version 0.01
 * Fall 2003
 *
 * Andrey Butov
 *
 *************************************************************************
******/

package wem;

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

/** Class responsible for displaying global variables. */
class WEMGlobalVariablePanel extends JPanel {
      /** Constructor. */
      WEMGlobalVariablePanel ( ) {
            setSize ( new Dimension(_width, _height) );
            setLayout ( new java.awt.GridLayout(0,1,5,0) );
            setBackground ( java.awt.Color.GREEN );
            setBorder ( new
javax.swing.border.LineBorder(java.awt.Color.BLACK) );

            _nameValuePairLabels = new LinkedList();
            _nameValuePairs = new LinkedList();

            initComponents ( );
      }

      void reset ( ) {
            _nameValuePairLabels.clear ( );
```

```
            _nameValuePairs.clear ( );
            removeAll();
            initComponents ( );
      }

      void updateRealTimeVariable ( String label, String v ) {
            WEMStringStringPair newPair = new WEMStringStringPair(label,
v);
            modifyExisting(newPair, true);
      }

      /** Initialize the components of this panel. */
      private void initComponents ( ) {
            JLabel titleLabel = new JLabel ( "GLOBAL VARIABLES" );
            titleLabel.setHorizontalAlignment (
javax.swing.SwingConstants.CENTER );
            add ( titleLabel );
      }

      /** Display a name and value pair */
      boolean displayNameValuePair ( WEMStringStringPair pair, boolean
isRealTime ) {
            if ( isRealTime == false && updateNeeded(pair) ) {
                  doDisplay ( pair, isRealTime );
                  return true;
            }
            return false;
      }

      /** Display a realtime variable. */
      void displayRealTimeVariable(WEMStringStringPair p) {
            doDisplay(p,true);
      }

      private boolean updateNeeded ( WEMStringStringPair pair ) {
            if ( pair == null ) return false;
            Iterator i = _nameValuePairs.iterator();
            while ( i.hasNext() ) {
                  WEMStringStringPair p = (WEMStringStringPair)i.next();
                  if ( p.getFirstString().equals(pair.getFirstString())
&&

p.getSecondString().equals(pair.getSecondString()) ) {
                        return false;
                  }
            }
            return true;
      }

      /** Finally display the name value pair. */
      private void doDisplay ( WEMStringStringPair pair, boolean
isRealTime ) {
            if ( modifyExisting(pair, isRealTime) == true ) return;
            JLabel finalLabel = new JLabel();
            finalLabel.setText ( formatLabel(pair, isRealTime) );
            finalLabel.setHorizontalAlignment (
javax.swing.SwingConstants.LEFT );
            _nameValuePairLabels.add ( new
WEMStringLabelPair(pair.getFirstString(), finalLabel) );
            _nameValuePairs.add(pair);
            add ( finalLabel );
      }

      /** Modify the value of an existing global variable. */
```

```
      private boolean modifyExisting ( WEMStringStringPair pair, boolean
isRealTime ) {
            Iterator i = _nameValuePairLabels.iterator();
            while ( i.hasNext() ) {
                  WEMStringLabelPair p = (WEMStringLabelPair)i.next();
                  if ( p.getString().equals(pair.getFirstString()) ) {
                        p.setLabelText ( formatLabel(pair, isRealTime)
);
                        return true;
                  }
            }
            return false;
      }

      /** Format the label as is appropriate to display a name and a
value. */
      private String formatLabel ( WEMStringStringPair data, boolean
isRealTime ) {
            if ( data == null ) return null;

            String labelString = new String();
            if ( isRealTime == true ) {
                  labelString = labelString.concat ( "REALTIME " );
            }

            labelString = labelString.concat ( data.getFirstString() );

            if ( (data.getSecondString().equals("")) == false ) {
                  labelString = labelString.concat ( " = " );
                  labelString = labelString.concat (
data.getSecondString() );
            }

            return labelString;
      }

      private final int _width = 100;
      private final int _height = 500;
      LinkedList _nameValuePairLabels;
      LinkedList _nameValuePairs;
}

/*************************************************************************
******
 *
 *  WEM - [W]hom [EM]ulator
 *  version 0.01
 *  Fall 2003
 *
 *  Andrey Butov
 *
 *************************************************************************
******/

package wem;

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

/**
 * The class has the responsibility of displaying log messages
 * such as errors and other notices sent to it during the execution
 * of WEM/WHOM source code.
 */
```

```
class WEMLogScreen extends JDialog {

      /** Main WEMLogScreen constructor. */
      WEMLogScreen ( ) {
            createViewableArea ( );
            setSize ( new Dimension(dialogWidth, dialogHeight) );
            setTitle ( "LOG" );
            // TODO: Set a nicer font.
            // TODO: Set a location relative to parent.
            getContentPane().add ( scrollPane, BorderLayout.CENTER );

      }

      /** Creates a viewable area for the log screen. */
      private void createViewableArea ( ) {
            // Create the text area.
            textArea = new JTextArea ( );
            textArea.setEditable ( false );

            // Stick the text area inside of a scroll pane.
            scrollPane = new JScrollPane ( textArea );
            scrollPane.setVerticalScrollBarPolicy (
JScrollPane.VERTICAL_SCROLLBAR_ALWAYS );
            scrollPane.setPreferredSize ( new Dimension(dialogWidth,
dialogHeight) );
      }

      /** Send a message to the log screen. */
      void log ( String s ) {
            append ( s );
      }

      /** Send a message to the log screen. */
      void append ( String s ) {
            textArea.append ( s );
            textArea.append ( "\n" );
      }

      /** Clear the log screen. */
      void clear ( ) {
            textArea.setText ( null );
      }

      private JTextArea textArea;
      private JScrollPane scrollPane;
      private final int dialogWidth = 750;
      private final int dialogHeight = 200;
}
/*********************************************************************
******
 *
 *  WEM - [W]hom [EM]ulator
 *  version 0.01
 *  Fall 2003
 *
 *  Andrey Butov
 *
 *********************************************************************
******/

package wem;

import java.awt.*;
import java.awt.event.*;
```

```
import java.io.*;
import javax.swing.*;
import javax.swing.filechooser.*;
import java.util.*;

/** This class represents the main WEM window. */
public class WEMMainWindow extends JFrame implements ActionListener,
WindowListener {

    /** The constructor for the main WEM window. */
    public WEMMainWindow ( WEM theApp ) {
        super ( mainWindowTitle );
        theApplication = theApp;
            whomInterface = theApplication.getInterface();

        // Preallocate some data.
        mainWindowDimension = new Dimension(mainWindowWidth,
mainWindowHeight);
            realTimeVariableMenuItems = new LinkedList();
            objectMenuItems = new LinkedList();
            objectPanels = new LinkedList();
            objectMenuItemActions = new LinkedList();

            // Deal with ongoing rimer.
            _timer = new java.util.Timer();
            _timerCandidates = new LinkedList();

            // Create the log screen here so that log messages are
            // received event if the log screen is not yet opened.
            logScreen = new WEMLogScreen ( );

        // Make sure the program terminates upon closing of this window.
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Set the window icon.
        setIconImage(getMainWindowIcon());

        // Set the size of the window.
        setSize(mainWindowDimension);

        // Make sure the window can't be resized.
        setResizable(false);

        // Add controls to the frame.
        createControls ( );

            // We're waiting for data from a WHOM source file.
            _dataDefinitionComplete = false;

            pack();
    }

    /** Reset after a new source file is chosen. */
    void reset ( ) {
            clearLogScreen();
            clearEnvironmentMenu();
            clearObjectPanels();
            clearGlobalVariablePanel();
            pack();
            _timer.cancel();
            _timer = new java.util.Timer();
            _timerStarted = false;
            _dataDefinitionComplete = false;
    }
```

```
      /** Signals the end of data definition coming in from the
interface. */
      void signalEndOfDataDefinition ( ) {
            _dataDefinitionComplete = true;
            startGlobalTimer ( );
      }

      /** Clear the log screen. */
      private void clearLogScreen ( ) {
            logScreen.clear();
      }

    /** Clear the ENVIRONMENT menu on the main menu. */
    private void clearEnvironmentMenu ( ) {
            resetRealTimeVariables();
            resetObjects();
    }

      /** Reset real-time variables state. */
      private void resetRealTimeVariables() {
            realTimeVariableMenu.removeAll();
            realTimeVariableMenuItems.clear();
      }

      /** Reset object states. */
      private void resetObjects() {
            objectMenu.removeAll();
            objectMenuItems.clear();
            objectMenuItemActions.clear();
      }

      /** Remove all displayed object panels. */
      private void clearObjectPanels() {
            Component[] components = getContentPane().getComponents();
            for ( int i = 0; i < components.length; i++ ) {
                  Component thisComponent = components[i];
                  if ( thisComponent instanceof WEMObjectPanel )
                        getContentPane().remove ( thisComponent );
            }
            objectPanels.clear();
      }

      private void clearGlobalVariablePanel ( ) {
                  _globalVariablePanel.reset();
                  pack();
      }

    /** Respond to various events. */
    public void actionPerformed ( ActionEvent event ) {
        Object source = event.getSource();
        if ( source == openMenuItem ) {
            File chosenFile = letUserOpenFile ( );
            if ( chosenFile != null ) {
                boolean ok = checkFileValidity (chosenFile);
                if ( ok == true ) {
                    informMainApplicationOfChosenFile ( chosenFile );
                    if (sourceViewer != null &&
sourceViewer.isShowing()) {
                        boolean loaded =
sourceViewer.loadDocument(chosenFile);
                        if ( loaded == false )
                            log ("Source Code Viewer: Error displaying
loaded file");
                    }
                }
```

```
            }
        }
            else if ( source == _reloadMenuItem ) {
                  informMainApplicationOfChosenFile (
theApplication.getCurrentlyLoadedSourceFile() );
            }
        else if ( source == exitMenuItem ) {
            System.exit ( 0 );
        }
        else if ( source == sourceViewerItem ) {
            displaySourceViewer();
            File f = theApplication.getCurrentlyLoadedSourceFile();
            if ( f != null ) {
                boolean loaded = sourceViewer.loadDocument(f);
                if ( loaded == false )
                    log("Error viewing loaded file");
            }
        }
            else if ( source == logScreenItem ) {
                  displayLogScreen();
            }
        else if ( source == aboutMenuItem ) {
            displayAboutDialog();
        }
            else if ( isRealTimeVariableMenuItem(source) ) {
                  JMenuItem item = (JMenuItem)source;
                  displayRealTimeVariablePanel ( item.getText() );
            }
            else if ( isObjectActionMenuItem(source) ) {
                  whomObjectActionToggle (
((WEMObjectActionMenuItem)source) );
            }
    }

      /** Send a message to the log screen. */
      void log ( String s ) {
            if ( logScreen != null ) logScreen.log ( s );
      }

    /**
     * Inform the main WEM application that a new WHOM source file
     * has been chosen.
     */
    void informMainApplicationOfChosenFile ( File chosenFile ) {
        if ( theApplication != null )
            theApplication.newSourceFileChosen ( chosenFile );
            else
                  log ( "WEM Error: theApplication = null" );
    }

    /** Returns the icon used for this window. */
    private Image getMainWindowIcon ( ) {
        java.net.URL imgURL =
WEMMainWindow.class.getResource(mainWindowIcon);
        return (imgURL != null) ? new ImageIcon(imgURL).getImage() :
null;
    }

    /** Create the controls of the main WEM window. */
    private void createControls ( ) {
            getContentPane().setLayout(new FlowLayout(FlowLayout.LEFT));
        createMainMenu ( );
            createGlobalVariablePanel ( );
    }
```

```
  /** Creates the global variable panel. */
  private void createGlobalVariablePanel ( ) {
        _globalVariablePanel = new WEMGlobalVariablePanel ( );
        getContentPane().add( _globalVariablePanel );
        pack ( );
  }

/** Create the main menu of this window. */
private void createMainMenu ( ) {
    mainMenuBar = new JMenuBar();
        mainMenuBar.add ( (JMenu)createFileMenu() );
        mainMenuBar.add ( (JMenu)createEnvironmentMenu() );
        mainMenuBar.add ( (JMenu)createToolsMenu() );
        mainMenuBar.add ( (JMenu)createHelpMenu() );
    setJMenuBar ( mainMenuBar );
  }

  /** Create the FILE menu on the main menu. */
  private JMenu createFileMenu ( ) {
        JMenu fileMenu = new JMenu("File");
        fileMenu.setMnemonic(KeyEvent.VK_F);

        _reloadMenuItem = new JMenuItem ( "Reload" );
        _reloadMenuItem.addActionListener ( this );
        fileMenu.add ( _reloadMenuItem );

    openMenuItem = new JMenuItem ( "Open" );
    openMenuItem.addActionListener ( this );
    fileMenu.add ( openMenuItem );

    exitMenuItem = new JMenuItem ( "Exit" );
    exitMenuItem.addActionListener ( this );
    fileMenu.add ( exitMenuItem );

        return fileMenu;
  }

  /** Create the ENVIRONMENT menu on the main menu bar. */
  private JMenu createEnvironmentMenu ( ) {
        JMenu environmentMenu = new JMenu ( "Environment" );
        environmentMenu.setMnemonic(KeyEvent.VK_E);

        realTimeVariableMenu = new JMenu ( "Real Time Variables" );

        objectMenu = new JMenu ( "Defined Objects" );

        environmentMenu.add ( realTimeVariableMenu );
        environmentMenu.add ( objectMenu );

        return environmentMenu;
  }

  /** Creates the TOOLS menu on the main menu bar. */
  private JMenu createToolsMenu ( ) {
        JMenu toolsMenu = new JMenu ( "Tools" );
    toolsMenu.setMnemonic(KeyEvent.VK_T);

        sourceViewerItem = new JMenuItem ( "Source Viewer" );
        sourceViewerItem.addActionListener ( this );
        toolsMenu.add ( sourceViewerItem );

        logScreenItem = new JMenuItem ( "Log Screen" );
        logScreenItem.addActionListener ( this );
        toolsMenu.add ( logScreenItem );
```

```
            return toolsMenu;
      }

      /** Creates the HELP menu on the main menu bar. */
      private JMenu createHelpMenu ( ) {
            JMenu helpMenu = new JMenu ( "Help" );
        helpMenu.setMnemonic(KeyEvent.VK_H);

            aboutMenuItem = new JMenuItem ( "About" );
        aboutMenuItem.addActionListener ( this );
        helpMenu.add ( aboutMenuItem );

            return helpMenu;
      }

    /** Respond to an error in validating a chosen WHOM source file. */
    void fileValidationError ( String s ) {
        String errorStr = "ERROR: ";
        log ( errorStr.concat(s) );
    }

      /** Responds to an error in parsing WHOM source code. */
      void whomError ( Exception e ) {
            String errorStr = "WHOM ERROR: ";
            errorStr = errorStr + e;
            log ( errorStr );
      }

    /**
     * Check if a given file is valid for WEM consumption.
     */
    private boolean checkFileValidity ( File f ) {
        try {
            if ( f.exists() == false ) {
                log ( "File does not exist" );
                return false;
            }
            else if ( f.canRead() == false ) {
                log ( "File cannot be read" );
                return false;
            }
            else if ( f.isFile() == false ) {
                log ( "Not a regular file" );
                return false;
            }
        }
        catch ( SecurityException e ) {
            log ( "WEMMainWindow.checkFileValidity(...) -
SecurityException" );
                  return false;
        }

        return true;
    }

    /** Let the user choose a WHOM source file. */
    private File letUserOpenFile() {
        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setFileSelectionMode (
JFileChooser.FILES_AND_DIRECTORIES );
            fileChooser.setCurrentDirectory(_currentDirectory);
            int returnVal =
fileChooser.showOpenDialog(WEMMainWindow.this);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            _currentDirectory = fileChooser.getCurrentDirectory();
```

```
                return fileChooser.getSelectedFile();
            }

        return null;
    }

    /** Display the WEM 'ABOUT' dialog. */
    private void displayAboutDialog ( ) {
            String[] s = { "[W]hom [EM]ulator", "version 0.01" };
            JOptionPane.showMessageDialog ( this, s, "About",
                                                                JOptionPane.
INFORMATION_MESSAGE);
    }

    /** Display the WHOM source code viewer. */
    private void displaySourceViewer ( ) {
        if (sourceViewer == null)
            sourceViewer = new WEMSourceViewer();
        if (sourceViewer.isShowing() == false)
            sourceViewer.setVisible(true);
        else
                sourceViewer.toFront();
    }

      /** Display the log screen. */
      private void displayLogScreen ( ) {
            if ( logScreen == null )
                  logScreen = new WEMLogScreen();
            if ( logScreen.isShowing() == false )
                  logScreen.setVisible(true);
            else
                  logScreen.toFront();
      }

    /** Responds to the window closed window event. */
    public void windowClosed(WindowEvent event) {
        Object source = event.getSource();
        if (source == sourceViewer) {
                sourceViewer.dispose();
                sourceViewer = null;
            }
    }

    /** Responds the the window opened window event. */
    public void windowOpened(WindowEvent event) {
            Object source = event.getSource();
        if (source == sourceViewer)
                displaySourceViewer();
      }

      /**
       * Incoming from the whom interface. The interface
       * will set all the realtime variables here.
       */
      void setAllRealTimeVariables ( LinkedList collection ) {
            Iterator i = collection.iterator();
            while ( i.hasNext() ) {
                  WEMStringStringPair p = (WEMStringStringPair)i.next();
                  JMenuItem item = new JMenuItem ( p.getFirstString() );
                  item.addActionListener(this);
                  realTimeVariableMenu.add(item);
                  realTimeVariableMenuItems.add(item);
                  if ( _globalVariablePanel != null ){
                        _globalVariablePanel.displayRealTimeVariable(p);
                  }
```

```
                String s = p.getFirstString();
                if ( s.equals("SECOND") ||
                       s.equals("MINUTE") ||
                       s.equals("HOUR") ) {
                      _timerCandidates.add ( p );
                }
          }
          pack ();
    }

    /**
     * Incoming from the whom interface. The interface
     * will set all the objects instances here.
     */
    void setAllObjects ( LinkedList collection ) {
          Iterator i = collection.iterator();
          while ( i.hasNext() ) {
                newObjectDefined ( (WEMModelObject)i.next() );
          }
          pack();
    }

    /**
     * Incoming from the interface. The interface will set
     * all the non-realtime global variables here.
     */
    void setAllGlobalVariables ( LinkedList collection ) {
          Iterator i = collection.iterator();
          while ( i.hasNext() ) {
                newGlobalVariableDefined((WEMStringStringPair)i.next()
, false);
          }
          pack();
    }

    /** An instance of a class has been defined. */
    void newObjectDefined ( WEMModelObject object ) {
          if ( displayNewObjectPanel ( object ) == false )
                return;

          JMenu newObjectMenu = new JMenu(object.getName());
          Iterator i = object.getActions().iterator();
          while ( i.hasNext() ) {
                WEMStringStringPair pair =
(WEMStringStringPair)i.next();
                JMenuItem actionItem =
                      new WEMObjectActionMenuItem(object.getName(),
                                                                  pair.g
etFirstString());

                objectMenuItemActions.add ( actionItem );
                actionItem.addActionListener(this);
                newObjectMenu.add(actionItem);
          }

          objectMenu.add ( newObjectMenu );
    }

    /* A new global variable is being defined. */
    void newGlobalVariableDefined ( WEMStringStringPair pair,
                                                      boolean isRealTime
) {
          if ( _globalVariablePanel == null ) return;
          _globalVariablePanel.displayNameValuePair ( pair, isRealTime
);
```

```
	}

	/** Display a new object panel. */
	private boolean displayNewObjectPanel ( WEMModelObject object ) {
		WEMObjectPanel existing = findObjectPanel(object.getName());
		if ( existing != null ) {
			existing.resolve(object);
			pack();
			return false;
		}
		else {
			WEMObjectPanel panel = new WEMObjectPanel(object);
			objectPanels.add(panel);
			getContentPane().add(panel);
			pack();
			return true;
		}
	}

	/** Is the given object one of the real-time variable menu items?
*/
	private boolean isRealTimeVariableMenuItem ( Object o ) {
		return realTimeVariableMenuItems.contains(o);
	}

	/**
	 * Is the given object an instance of one of the menu items
responsible
	 * for an object action?
	 */
	private boolean isObjectActionMenuItem ( Object o ) {
		return objectMenuItemActions.contains(o);
	}

	/** Is the given object one of the object menu items? */
	private boolean isObjectMenuItem ( Object o ) {
		return objectMenuItems.contains(o);
	}

	/** Display a panel to let the user modify a real-time variable.
*/
	private void displayRealTimeVariablePanel ( String label ) {
		String[] info = { label, " ", "Modification?" };
		String retVal = JOptionPane.showInputDialog(this, info,
label,
			JOptionPane.QUESTION_MESSAGE);

		if ( retVal == null )
			return;

		/*************** OBSOLETE
*****************************************
		Double d;
		try { d = new Double(retVal); }
		catch ( NumberFormatException e ) {
			log( "ERROR: Invalid Entry...Changes Not Applied." );
			return;
		}
		************************************************************
**********/

		doUpdateRealTimeVariable ( label, retVal );
	}
```

```
	/** Given the new value for a real time variable, reflect the new
value
	 * in the GUI, and send the new value to the backend.
	 */
	void doUpdateRealTimeVariable ( String label, String newValue ) {
		_globalVariablePanel.updateRealTimeVariable ( label,
newValue );
		pack();
		whomInterface.OUTGOING_RealTimeVariableChanged(label,
newValue);
		pack();
	}

	/**
	 * A menu item designating an attribute of a defined WHOM object
	 * was toggled.
	 */
	private void whomObjectActionToggle ( WEMObjectActionMenuItem
menuItem ) {
		String actionName = menuItem.getText();
		String objectName = menuItem.getObjectName();

		WEMObjectPanel panel = findObjectPanel(objectName);

		if ( panel != null )
			panel.toggle ( actionName );

		pack();

		whomInterface.OUTGOING_ObjectActionActivated(objectName,
actionName);
	}

	/** Finds a WEMObjectPanel in this GUI given the name of the
object. */
	private WEMObjectPanel findObjectPanel ( String objectName ) {
		Iterator i = objectPanels.iterator();
		while ( i.hasNext() ) {
			WEMObjectPanel wop = (WEMObjectPanel)i.next();
			if ( wop.getObjectName().equals(objectName) ) {
				return wop;
			}
		}
		return null;
	}

	/**
	 * Registers a timer to take care of continuously updating
	 * real-time variables. Currently these include
	 * HOUR, MINUTE and SECOND.
	 */
	private void startGlobalTimer ( ) {
		if ( _timerStarted || !(_dataDefinitionComplete) )
			return;

		// Schedule a timer to check the time every second.
		// If second, minute, or hour has been modified,
		// update the appropriate realtime variables, if they
		// have been defined.

		_timer.scheduleAtFixedRate ( new
WEMTimerTask(_timerCandidates, this), 0, 1000 );
		_timerStarted = true;
	}
```

```
      // Ignored interface methods.
    public void windowClosing(WindowEvent e){}
    public void windowDeactivated(WindowEvent e){}
    public void windowDeiconified(WindowEvent e){}
    public void windowIconified(WindowEvent e){}
    public void windowActivated(WindowEvent e){}

    // Members dealing with the default properties of the main window.
    private static final String mainWindowTitle = "WEM - [W]hom
[EM]ulator";
    private final String mainWindowIcon = "images/houseicon.gif";
    private final int mainWindowWidth = 1000;
    private final int mainWindowHeight = 700;
    private final Dimension mainWindowDimension;

    // Various menu items.
    private JMenuBar mainMenuBar;                              // The main
menu bar.
      private JMenu realTimeVariableMenu;                      //
Environment->Real Time Variables
      private JMenu objectMenu;                                      //
Environment->Objects
      private JMenuItem _reloadMenuItem;                       // File-
>Reload
      private JMenuItem openMenuItem;                          // File-
>Open
    private JMenuItem exitMenuItem;                     // File-Exit
    private JMenuItem sourceViewerItem;                 // Tools->Source
Viewer
      private JMenuItem logScreenItem;                         // Tools-
>Log Screen
    private JMenuItem aboutMenuItem;                    // Help->About
    private LinkedList realTimeVariableMenuItems;
      private LinkedList objectMenuItems;
      private LinkedList objectMenuItemActions;
      private LinkedList objectPanels;
      private WEMGlobalVariablePanel _globalVariablePanel;

    // Various child windows.
    private WEMSourceViewer sourceViewer;
      private WEMLogScreen logScreen;

    // The main WEM application which owns this window.
    private WEM theApplication;
      private WEMWhomInterface whomInterface;

      // Various data.
      private boolean _dataDefinitionComplete;
      private java.util.Timer _timer;
      private LinkedList _timerCandidates;
      private boolean _timerStarted = false;

      // Data related to the chosen WHOM source file.
      File _currentDirectory;
}

package wem;

import java.util.*;

/** Class modeling a generic WHOM object. */
class WEMModelObject {
      WEMModelObject ( String name ) { setName ( name ); }

      String getName ( ) { return objectName; }
```

```
      ArrayList getActions ( ) { return objectActions; }
      ArrayList getAttributes ( ) { return objectAttributes; }

      void setName ( String name ) { objectName = name; }
      void setActions ( ArrayList a ) { objectActions = a; }
      void setAttributes ( ArrayList a ) { objectAttributes = a; }

      private String objectName;
      private ArrayList objectActions;
      private ArrayList objectAttributes;
}

package wem;

/**
 * Class modeling a generic WHOM realtime variable.
 */
class WEMModelRealTimeVariable {
    // TODO: What is this about?
      //whom.backend.ObservableObject oo = null;

      /** Constructor */
      WEMModelRealTimeVariable(String name) {
            setName ( name );
      }

      /** Returns the varible name. */
      String getName() {
            return variableName;
      }

      /** Set the variable name. */
      void setName ( String name ) {
            variableName = name;
      }

      private String variableName;
}
/************************************************************************
******
 *
 *  WEM - [W]hom [EM]ulator
 *  version 0.01
 *  Fall 2003
 *
 *  Andrey Butov
 *
 ************************************************************************
******/

package wem;

/**
 * The class represents one action of a WHOM ojbect represented
 * as a JMenuItem.
 */
class WEMObjectActionMenuItem extends javax.swing.JMenuItem {

      /**
       * Constructor.
       */
      WEMObjectActionMenuItem ( String objectName, String label ) {
            super ( label );
            _objectName = objectName;
      }
```

```
      /**
       * Returns the name of the object which suppports this action.
       */
      String getObjectName ( ) {
            return _objectName;
      }

      private String _objectName;
}
/**********************************************************************
******
 *
 *  WEM - [W]hom [EM]ulator
 *  version 0.01
 *  Fall 2003
 *
 *  Andrey Butov
 *
 **********************************************************************
******/

package wem;

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

/**
 * Class responsible for presenting a panel with information regarding
 * a WEMModelObject.
 */
class WEMObjectPanel extends JPanel {
      WEMObjectPanel ( WEMModelObject object ) {
            setSize ( new Dimension(width, height) );

            objectName = new JLabel(object.getName());
            objectName.setHorizontalAlignment(javax.swing.SwingConstants
.CENTER);

            objectActions = object.getActions();
            objectAttributes = object.getAttributes();

            objectActionLabels = new ArrayList();
            objectAttributeLabels = new ArrayList();

            initComponents ( );
      }

      void resolve ( WEMModelObject incoming ) {
            resolveActions(incoming.getActions());
            resolveAttributes(incoming.getAttributes());
      }

      private void resolveActions ( ArrayList actions ) {
            Iterator i = actions.iterator();
            while ( i.hasNext() ) {
                  WEMStringStringPair incomingPair =
(WEMStringStringPair)i.next();
                  WEMStringStringPair existing =
findExistingAction(incomingPair.getFirstString());
                  if ( existing != null ) {
                        existing.setSecondString(incomingPair.getSecondS
tring());
```

```
                        updateActionLabel ( existing.getFirstString(),
existing.getSecondString() );
                  }
            }
      }

      private void resolveAttributes ( ArrayList attributes ) {
            Iterator i = attributes.iterator();
            while ( i.hasNext() ) {
                  WEMStringStringPair incomingPair =
(WEMStringStringPair)i.next();
                  WEMStringStringPair existing =
findExistingAttribute(incomingPair.getFirstString());
                  if ( existing != null ) {
                        existing.setSecondString(incomingPair.getSecondS
tring());
                        updateAttributeLabel (
existing.getFirstString(), existing.getSecondString() );
                  }
            }
      }

      private void updateActionLabel ( String name, String newValue ) {
            if ( objectActionLabels == null )
                  return;

            Iterator i = objectActionLabels.iterator();
            while ( i.hasNext() ) {
                  WEMStringLabelPair p = (WEMStringLabelPair)i.next();
                  if ( p.getString().equals(name) ) {
                        p.setLabelText(formatLabel(new
WEMStringStringPair(name,newValue)));
                        break;
                  }
            }
      }

      private void updateAttributeLabel ( String name, String newValue )
{
            if ( objectAttributeLabels == null )
                  return;

            Iterator i = objectAttributeLabels.iterator();
            while ( i.hasNext() ) {
                  WEMStringLabelPair p = (WEMStringLabelPair)i.next();
                  if ( p.getString().equals(name) ) {
                        p.setLabelText(formatLabel(new
WEMStringStringPair(name, newValue)));
                        break;
                  }
            }
      }

      WEMStringStringPair findExistingAction ( String name ) {
            Iterator i = objectActions.iterator();
            while ( i.hasNext() ) {
                  WEMStringStringPair p = (WEMStringStringPair)i.next();
                  if ( p.getFirstString().equals(name) )
                        return p;
            }
            return null;
      }

      WEMStringStringPair findExistingAttribute ( String name ) {
            Iterator i = objectAttributes.iterator();
```

```
            while ( i.hasNext() ) {
                  WEMStringStringPair p = (WEMStringStringPair)i.next();
                  if ( p.getFirstString().equals(name) )
                        return p;
            }
            return null;
      }

      /** Returns the object label name. */
      String getObjectName ( ) { return objectName.getText(); }

      /** Toggle the state of an action for this object. */
      void toggle ( String actionName ) {
            Iterator i = objectActionLabels.iterator();
            while ( i.hasNext() ) {
                  WEMStringLabelPair p = (WEMStringLabelPair)i.next();
                  if ( p.getString().equals(actionName) ) {
                        if (
p.getLabel().getForeground().equals(java.awt.Color.GREEN) )
                              p.getLabel().setForeground(java.awt.Color.
BLACK);
                        else
                              p.getLabel().setForeground(java.awt.Color.
GREEN);
                        break;
                  }
            }
      }

      private void initObjectActionComponents ( ) {
            if ( objectActions == null )
                  return;

            Iterator i = objectActions.iterator();
            if ( i == null )
                  return;

            while ( i.hasNext() )
            {
                  WEMStringStringPair pair =
(WEMStringStringPair)i.next();
                  if ( pair == null )
                        continue;

                  JLabel l = new JLabel(formatLabel(pair));
                  l.setBackground(java.awt.Color.WHITE);
                  l.setHorizontalAlignment(javax.swing.SwingConstants.LE
FT);
                  objectActionLabels.add(new
WEMStringLabelPair(pair.getFirstString(),l));
                  add ( l );
            }
      }

      private void initObjectAttributeComponents ( ) {
            if ( objectAttributes == null ) return;

            Iterator i = objectAttributes.iterator();
            if ( i == null ) return;

            while ( i.hasNext() ) {
                  WEMStringStringPair pair =
(WEMStringStringPair)i.next();
                  if ( pair == null ) continue;
```

```
                  JLabel l = new JLabel(formatLabel(pair));
                  l.setHorizontalAlignment(javax.swing.SwingConstants.LE
FT);
                  objectAttributeLabels.add(new
WEMStringLabelPair(pair.getFirstString(), l));
                  add ( l );
            }
      }

      /** Initialize components. */
      private void initComponents ( ) {
            setLayout ( new java.awt.GridLayout(0,1,5,0) );
            setBackground ( new java.awt.Color(153,153,255) );
            setBorder ( new javax.swing.border.LineBorder(new
java.awt.Color(0,0,0)) );
            add ( objectName );

            initObjectActionComponents ( );
            initObjectAttributeComponents ( );
      }

      /** Format the text of a label given a pair of strings for name
and value. */
      private String formatLabel ( WEMStringStringPair p ) {
            String r = p.getFirstString();
            if ( p.getSecondString() != null &&
(p.getSecondString().equals("")) == false ) {
                  r = r.concat(" = " );
                  r = r.concat(p.getSecondString());
            }
            return r;
      }

      private final int       width = 100;
      private final int       height = 500;
      private JLabel          objectName;
      private     ArrayList   objectActions;
      private ArrayList objectActionLabels;
      private ArrayList objectAttributes;
      private ArrayList objectAttributeLabels;
}
/*********************************************************************
******
 *
 *  WEM - [W]hom [EM]ulator
 *  version 0.01
 *  Fall 2003
 *
 *  Andrey Butov
 *
 *********************************************************************
******/

package wem;

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

/**
 * The class has the responsibility of displaying a given
 * text file. The main WEM application will utilize this class
 * to display the user-chosen WHOM source files.
 */
class WEMSourceViewer extends JDialog {
```

```
    /**
     * Main WEMSourceViewer constructor.
     */
    WEMSourceViewer ( ) {
        createViewableArea();
        setSize(new Dimension ( dialogWidth, dialogHeight ));
            // TODO: Set a nicer font.
        // TODO - set location relative to parent.
        getContentPane().add ( scrollPane, BorderLayout.CENTER );
    }

    /**
     * Load a file into the source viewer.
     */
    boolean loadDocument ( File f ) {
        try {
            if ( f != null ) {
                textArea.setText(null); // clear the old text...
                StringBuffer sb = new StringBuffer();
                FileReader reader = new FileReader(f);
                int c;
                for ( ;; ) {
                    c = reader.read();
                    if ( c == -1 )
                        break;
                    else
                        sb.append((char)c);
                }

                setTitle(f.getName());
                textArea.setText(sb.toString());
                return true;
            }
        }
        catch ( FileNotFoundException e ) {
            System.err.println("FileNotFoundException: "+e);
                  // TODO - deal with this better.
        }
        catch ( IOException e ) {
                  System.err.println("IOException: "+e);
            // TODO - deal with this better.
        }

        return false;
    }

    /**
     * Create the area in which the text will be displayed.
     */
    private void createViewableArea ( )
    {
        // Create the text area...
        textArea = new JTextArea ( );
        textArea.setEditable(false);

        // Stick the text area inside of a scroll pane.
        scrollPane = new JScrollPane(textArea);
        scrollPane.setVerticalScrollBarPolicy (
                                        JScrollPane.VERTICAL_SCROLLBAR_AL
WAYS );

        scrollPane.setPreferredSize(new Dimension(dialogWidth,
dialogHeight));
    }
```

```
    private JTextArea textArea;
    private JScrollPane scrollPane;
    private final int dialogWidth = 400;
    private final int dialogHeight = 500;
}

package wem;

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

class WEMStringLabelPair {
      WEMStringLabelPair ( String s, JLabel l ) {
            _string = s;
            _label = l;
      }

      String getString ( ) { return _string; }
      JLabel getLabel ( ) { return _label; }

      void setLabelText ( String text ) {
            if ( _label == null ) return;
            _label.setText ( text );
      }

      private String _string;
      private JLabel _label;
}

package wem;

/** Class encapsulating a pair consisting of a String and a String. */
class WEMStringStringPair {

      WEMStringStringPair ( String s1, String s2) {
            _string1 = s1;
            _string2 = s2;
      }

      WEMStringStringPair ( String s1, double d ) {
            this ( s1, java.lang.Double.toString(d) );
      }

      WEMStringStringPair ( String s1, boolean b ) {
            this ( s1, java.lang.Boolean.toString(b) );
      }

      /** Returns the first string. */
      String getFirstString ( ) { return _string1; }

      /** Returns the second string. */
      String getSecondString ( ) { return _string2; }

      /** Sets the first string. */
      void setFirstString ( String s ) { _string1 = s; }

      /** Sets the second string. */
      void setSecondString ( String s ) { _string2 = s; }

      private String _string1;
      private String _string2;
}
/*********************************************************************
******
```

```
 *
 *  WEM - [W]hom [EM]ulator
 *  version 0.01
 *  Fall 2003
 *
 *  Andrey Butov
 *
 ***************************************************************************
******/

package wem;

import java.util.*;

class WEMTimerTask extends TimerTask {

      /** Constructor. */
      WEMTimerTask ( LinkedList realtimeVariables, WEMMainWindow window
) {
            _window = window;

            Iterator i = realtimeVariables.iterator();
            while ( i.hasNext() ) {
                  WEMStringStringPair p = (WEMStringStringPair)i.next();
                  String fs = p.getFirstString();
                  if ( fs.equals("HOUR") )
                        _hour = p;
                  else if ( fs.equals("MINUTE") )
                        _minute = p;
                  else if ( fs.equals("SECOND") )
                        _second = p;
            }
      }

      public void run ( ) {
            if ( _second != null ) {
                  int newVal = (new
GregorianCalendar().get(Calendar.SECOND));
                  if ( newVal != _lastSecond ) {
                        _second.setSecondString((new
Double(newVal)).toString());
                        _window.doUpdateRealTimeVariable(_second.getFirs
tString(),

       new String(Integer.toString(newVal)));
                        _lastSecond = newVal;
                  }
            }
            if ( _minute != null ) {
                  int newMin = (new
GregorianCalendar().get(Calendar.MINUTE));
                  if ( newMin != _lastMinute ) {
                        _minute.setSecondString((new
Double(newMin)).toString());
                        _window.doUpdateRealTimeVariable(_minute.getFirs
tString(),

      new String(Integer.toString(newMin)));
                        _lastMinute = newMin;
                  }
            }
            if ( _hour != null ) {
                  int newHour = (new
GregorianCalendar().get(Calendar.HOUR));
                  if ( newHour != _lastHour ) {
```

```
                        _minute.setSecondString((new
Double(newHour)).toString());
                        _window.doUpdateRealTimeVariable(_hour.getFirstS
tring(),

      new String(Integer.toString(newHour)));
                        _lastHour = newHour;
                  }
            }
      }

      private WEMStringStringPair _hour = null;
      private WEMStringStringPair _minute = null;
      private WEMStringStringPair _second = null;
      private int _lastSecond = -1;
      private int _lastMinute = -1;
      private int _lastHour = -1;
      private WEMMainWindow _window;
      private Calendar _calendar;
}
/*************************************************************************
******
 *
 *  WEM - [W]hom [EM]ulator
 *  version 0.01
 *  Fall 2003
 *
 *  Andrey Butov
 *
 *************************************************************************
******/

package wem;

import java.io.*;
import java.util.*;
import whom.*;
import whom.backend.*;
import antlr.CommonAST;

/**
 * Class responsible for interfacing with the WHOM interpreter.
 * This is the only WEM class exposed to the Backend of the WHOM
interpreter.
 */
public class WEMWhomInterface {

      /* Constructor */
      WEMWhomInterface(WEM application) {
            app = application;

            // The following MUST be called before any Backend
operations.
            Backend.wemInterface = this;
      }

      /* Send a message to the WEM log screen. */
      public void BACKEND_INTERFACE_Log ( String s ) {
            log ( s );
      }

      /** Start the interface with a new WHOM source file. */
      void start(File f) throws Exception {
            Reader reader = new FileReader(f);
            startWhomPipeline(reader);
```

```
	}

	/** Starts the main WHOM interpreter pipeline. */
	private void startWhomPipeline ( Reader r ) throws Exception {
		WHOMParser parser = new WHOMParser(new WHOMLexer(r));
		parser.program();
		CommonAST tree = (CommonAST)parser.getAST();
		Backend.init();
		WHOMWalker walker = new WHOMWalker();
		walker.program(tree);

		_realTimeVariablesDefined = false;
		queryAllDefinedData();
		app.getMainWindow().signalEndOfDataDefinition();
	}

	/** Get all the defined objects from within the source code. */
	private void queryAllDefinedData() {
		 _highestScope = InstantiatedScope.current();

		// Real time variables cannot be changed from within the
WHOM
		// source code. They are typically updated through
environmental
		// conditions, and therefore are only changeable through the
WEM application.

		if ( _realTimeVariablesDefined == false )
			queryRealTimeVariables();

		queryGlobalVariables();
		queryObjects();
	}

	/**
	 * Create a WEMModelObject object given a whom.backend.Object
	 * and the object name.
	 */
	WEMModelObject createWEMModelObject ( String name ) {
		whom.backend.Object object =
(whom.backend.Object)_highestScope.getObject(name);
		WEMModelObject returnable = new WEMModelObject(name);
		ArrayList objectActions = new ArrayList();
		ArrayList objectAttributes = new ArrayList();

		Collection attributenames =
object.getClassRef().lookupAllNames();
		Iterator i = attributenames.iterator();
		while ( i.hasNext() ) {
			String attributename = (String) i.next();
			whom.backend.Attribute a =
object.getClassRef().lookupAttribute(attributename);
			switch ( a.getType() ) {
				case whom.backend.Type.Boolean:
					whom.backend.Boolean b =
(whom.backend.Boolean)object.get(attributename);
					objectActions.add ( new
WEMStringStringPair(attributename, b.getValue()) );
					break;
				case whom.backend.Type.Number:
					whom.backend.Number n =
(whom.backend.Number)object.get(attributename);
					objectAttributes.add ( new
WEMStringStringPair(attributename, n.getNumber()) );
					break;
```

```
                        case whom.backend.Type.String:
                              whom.backend.WhomString s =
(whom.backend.WhomString)object.get(attributename);
                              objectAttributes.add ( new
WEMStringStringPair(attributename, s.getString()) );
                              break;
                        case whom.backend.Type.Object:
                              whom.backend.Object o =
(whom.backend.Object)object.get(attributename);
                              objectAttributes.add ( new
WEMStringStringPair(attributename, null) );
                              break;
                        case whom.backend.Type.Void:
                        default:
                              log ("What kind of object is this???");
                              continue;
                  }
            }

            returnable.setActions ( objectActions );
            returnable.setAttributes ( objectAttributes );

            return returnable;
      }

      /** Called when a realtime variable has been modified in WEM. */
      void OUTGOING_RealTimeVariableChanged(String name, String value) {
            ObservableObject oo = _highestScope.getVariable(name);
            if ( oo != null ) {
                  if ( isRealTimeNumber(oo) ) {
                        whom.backend.Number n = (whom.backend.Number)oo;

                        double d;
                        try {
                              d = Double.parseDouble(value);
                        }
                        catch ( NumberFormatException e ) {
                              log ( e.toString() );
                              return;
                        }

                        n.setNumber ( d );
                  }
                  else if ( isRealTimeString(oo) ) {
                        whom.backend.WhomString s =
(whom.backend.WhomString)oo;
                        s.setString ( value );
                  }
            }

            ObservableObject.processEvents();
            queryAllDefinedData();
      }

      /** Called when a non-realtime object had one of its actions
activated. */
      void OUTGOING_ObjectActionActivated(String objectName, String
actionName) {
            whom.backend.Object whomObject =
_highestScope.getObject(objectName);
            if ( whomObject == null ) {
                  log ( "SEVERE: Failed to find object during action
activation." );
                  return;
            }
```

```
            whom.backend.Boolean b_on =
(whom.backend.Boolean)whomObject.get(actionName);

            if ( b_on.getValue() == true ) b_on.setValue(false);
            else b_on.setValue(true);

            ObservableObject.processEvents();
            queryAllDefinedData();
      }

      /** Returns whether or not a given observable object is a realtime
number. */
      private boolean isRealTimeNumber(ObservableObject oo) {
            return ( oo instanceof whom.backend.Number &&
oo.isObservable() );
      }

      /** Returns whether or not a given observable object is a realtime
string. */
      private boolean isRealTimeString(ObservableObject oo) {
            return ( oo instanceof whom.backend.WhomString &&
oo.isObservable() );
      }

      /** Send a message to the WEM log screen. */
      private void log ( String s ) {
            app.getMainWindow().log ( s );
      }

      /** Query all real-time variables from the backend. */
      private void queryRealTimeVariables ( ) {
            LinkedList collection = new LinkedList();

            Collection allNames = _highestScope.allNames();
            Iterator i = allNames.iterator();

            while ( i.hasNext() ) {
                  String name = (String)i.next();
                  ObservableObject oo = _highestScope.getVariable(name);
                  if ( oo != null ) {
                        if ( isRealTimeNumber(oo) ) {
                              whom.backend.Number number =
(whom.backend.Number)oo;
                              collection.add(new
WEMStringStringPair(name, number.getNumber()));
                        }
                        else if ( isRealTimeString(oo) ) {
                              whom.backend.WhomString str =
(whom.backend.WhomString)oo;
                              collection.add(new
WEMStringStringPair(name, str.getString()));
                        }
                  }
            }

            _realTimeVariablesDefined = true;
            app.getMainWindow().setAllRealTimeVariables(collection);
      }

      /** Query the existence and state of all global variables. */
      private void queryGlobalVariables ( ){
            LinkedList collection = new LinkedList();

            Collection allNames = _highestScope.allNames();
```

```
            Iterator i = allNames.iterator();

            while ( i.hasNext() ) {
                  String name = (String)i.next();
                  ObservableObject oo = _highestScope.getVariable(name);
                  if ( oo != null && isRealTimeNumber(oo) == false &&
isRealTimeString(oo) == false) {
                        switch ( oo.getType() ) {
                              case whom.backend.Type.Object:
                              case whom.backend.Type.Unknown:
                              case whom.backend.Type.Void:
                              case whom.backend.Type.Boolean:
                                    break;
                              case whom.backend.Type.Number:
                                    whom.backend.Number n =
(whom.backend.Number)oo;
                                    collection.add(new
WEMStringStringPair(name, n.getNumber()));
                                    break;
                              case whom.backend.Type.String:
                                    whom.backend.WhomString s =
(whom.backend.WhomString)oo;
                                    collection.add(new
WEMStringStringPair(name, s.getString()));
                                    break;
                        }
                  }
            }

            app.getMainWindow().setAllGlobalVariables(collection);
      }

      private void queryObjects ( ) {
            LinkedList collection = new LinkedList();

            Collection allNames = _highestScope.allObjectsNames();
            Iterator i = allNames.iterator();

            while ( i.hasNext() ) {
                  String name = (String)i.next();
                  collection.add ( createWEMModelObject(name) );
            }

            app.getMainWindow().setAllObjects ( collection );
      }

      private WEM app;
      private InstantiatedScope _highestScope;
      private boolean _realTimeVariablesDefined = false;
}
```

Testing files

Level 1

```
//whitespace1.whom

//whitespace2.whom




//whitespace_comments1.whom
// This should be ignored completely!

//    As should this
```

```
//
// ... and this ...
//

//whitespace_comments2.whom

/* This should be ignored... */

/***************** As should this ***********/


/*
 *
 *   AND THIS
 *
 */


//    /* This as well */


/*    // And this */

//    /* The missing comment closure should not even be seen...

/*

     Ignore me completely....I'm not even HERE!


*/
```

Level 2

```
/* Forced_error1.whom
 * Tests the WHOM pipeline by forcing an error. See note below
 * for explanation.
 */

realtime number realTimeNumber1 = 1;
realtime number SECOND;

number nonRealTimeNumber;

class C
{
      EVENT_ONE;
      number memberNum = 100;
}

C someObjectOne;

// THE FOLLOWING GIVES AN ERROR...OBVISOULY!,
// THE COMPILATION SHOULD FAIL COMPLETELY, AND A SIGNAL
// INDICATING THIS SHOULD BE SENT TO THE BACKEND, SO THAT
// WEM WILL STOP ALL OPERATIONS.

once ( SECOND == 2 ) {
      AaBbCcDdEeFf = nonRealTimeNumber + 1;
}

// THE FOLLOWING WORKS FINE.
once ( SECOND == 15 ) {
      nonRealTimeNumber = nonRealTimeNumber + 1;
}
```

```
/*
 * loop_if_else_test1.whom
 * Tests the if-else, while and for loops.
 */

realtime number i;
number b=1;

class foo
{
    number bar=0;
}

foo foo1;

once (i)
{
    foo1.bar=2;
    number c=0;
    for(c=0;c<10;c=c+1)
      {
            foo1.bar=foo1.bar+1;
    }

    while(i)
      {
            foo1.bar=foo1.bar-1;
            i=i-1;
    }

    if( i<=0 )
      {
            foo1.bar = foo1.bar*7;
    }
    else
      {
            foo1.bar = foo1.bar*3;
    }
}

/* parsing_realtime_vars1.whom*/
/* this example simply tests the creation and parsing of realtime
variables. */

realtime number A;
realtime number B;
realtime number C = 1;
realtime number D = -1;
realtime number E = 999999999;
realtime number F = -999999999;

/*
 * string_test1.whom
 * A simple attempt to display string values in WEM.
 */

realtime number SECOND;
string s;

class Foo {
   void bar ( ) {
            s = "hello";
   }
   void clear ( ) {
```

```
         s = "goodbye";
   }
}

Foo f;

once ( SECOND >= 30 ) {
      f.bar();
}

once ( SECOND < 30 ) {
      f.clear();
}


/*
 * once_statement_test1.whom
 * Test the usage of an expression as the predicate for the
 * once() statement.
 */

realtime number i;
number b=1;

once ((b+1)<i)
{
  b=i-b;
  b=b+b;
  b=b+i;
  b=b+b;
}
```

Level 3

```
/*
 * event_dependency1.whom
 * Test if a global non-realtime number responds to a
 * change in the realtime variable triggered by an event.
 */

realtime number i;
number b=1;

once (i)
{
  b=i;
}

/*
 * event_dependency2.whom
 * Test the modification of a non-realtime global variable
 * based on a change in a real time number triggered by an event.
 */

realtime number i;
number b=1;

once (i)
{
  b=i-b;
  b=b+b;
  b=b+i;
  b=b+b;
}

/**
```

```
 * event_logic1.whom
 * Test the modification of a realtime variable inside an
 * event trigerred by a change in the same variable.
 */

realtime number i;
number b=1;

once (i)
{
  b=i;
  i=i-1;
}

/*
 * event_logic2.whom
 * Test of a modification of a realtime variable inside an
 * event triggered by the modification of the same variable.
 */

realtime number i;
number b=1;

once (i)
{ /* b is 2 raised to the power i */
  b=b*2;
  i=i-1;
}
/**
 * event_logic3.whom
 * Testing the trigerring of two executions of an event.
 */

realtime number i=0;

number b=0;

once(i>4) {
    // testing if this triggers two executions of the event
    i=i-1;
    i=i-1;
}

once(i>-10) {
    b=b+1;
}

/**
 * event_logic4.whom
 * Tests event logic by having various predicates based
 * off of the same class member: Light::EVENT_ON.
 */

realtime number i;

class Light
{
      EVENT_ON;
      number level=1;
}

Light l1;

once (i)
{
```

```
      l1.level=l1.level*2;
}

once  l1.EVENT_ON
{
      l1.level=l1.level*3;
}

once (!l1.EVENT_ON)
{
      l1.level=l1.level*5;
}

/*
 * example_for_prof.whom
 * Test the usage of an expression as the predicate for the
 * once() statement.
 */

import "whom/lib/whom.wl";

Door door1;

SecurityCam scam;

Alarm al;

// take a picture every 15 seconds
once ( (SECOND == 0) | (SECOND == 15) | (SECOND == 30) | (SECOND == 45)
)
{
    scam.takePicture();
}

once ( door1.EVENT_KNOCKED )
{
    scam.takePicture();
    // make sure they can knock again
    door1.EVENT_KNOCKED = false;
}

// somebody tries to break in
once ( door1.EVENT_OPEN_INSIDE & door1.EVENT_LOCKED )
{
    scam.takePicture();
    al.trigger();
}
```

Level 4

```
/* recursion.whom */
// let's try to compute the Fibonacci numbers recursively

// the number that is observed
realtime number i=0;

// our standard test number
number b=0;

class foo {
    number fib(number x) {
      // this is ineffective by purpose ;-)
      if( x == 0 )
          return 0;
      else if( x == 1)
```

```
            return 1;
        else
            return fib(x-1) + fib(x-2);
    }
}

foo foo1;

once(i) {
    b = foo1.fib(i);
}

// Defines realtime TIME related variables
// Test if WEM appropriately deals with these.

/* time1.whom*/

realtime number SECOND;
realtime number MINUTE;
realtime number HOUR;

/* time2.whom */
// After adding some realtime TIME related variables,
// See if events respond to changes related to time.

realtime number SECOND;
realtime number MINUTE;
realtime number HOUR;

number b = 1;

once ( SECOND > 1 )
{
      b = b+1;
}
```

Misc

```
/*
 * include_standard_whom_library.whom
 * whom.wl must be in current directory...or
 * use an explicit path.
 */

import "whom.wl";

/*
 * ... code here ...
 */


/*
 * milestone2_2.whom
 * WHOM project milestone program 2-2. An extensive test of
 * object declaration but group, and, and all operations
 * are not included.
*/

import "testing/whom/misc/milestone2_1.wl";

Light mainLight;
Lamp myLamp;
Door myDoor;
Room myRoom;
```

```
put (mainLight, myLamp, myDoor) in myRoom;

once (TEMPERATURE>100 & HOUR==12)
{
      myDoor.open();
}

once ( HOUR > 23 )
{
      myLamp.status = 0;
      mainLight.status = 0;
}

once (myDoor.EVENT_OPEN_OUTSIDE & (HOUR > 18 | HOUR < 6 ) )
{
      myLamp.status = 1;
      mainLight.status = 1;
}

/*
 * milestone2_1.whom
 * WHOM project milestone library 2.1
 * Purpose: This file defines two classes with
 * events, attributes, and functions in order to
 * test the class definition part.
 */

realtime number TEMPERATURE;
realtime number HOUR;
realtime number LIGHTNESS;

class Room
{
}

class Light
{
      //number status = 0;
      //built-in events, the control system will notify WHOM
      //when this happens

      EVENT_TURNON;
      EVENT_TURNOFF;
}

class Lamp extends Light
{
    number light_level;

      //number status = 0;
      number SelfAdjust()
      {
            // adjusts light level based on lightness
            light_level = (LIGHTNESS/10.0)*3.0;
            return light_level;
      }
}

class Door
{
      number status = 0;
      // built in events, the control system will notify WHOM
      // when this happens
      EVENT_OPEN_OUTSIDE;
      EVENT_OPEN_INSIDE;
```

```
	EVENT_CLOSE_INSIDE;
	EVENT_CLOSE_OUTSIDE;
	void open()
	{
		status=0;
	}
	void close()
	{
		status=1;
	}
}

/*
 * original_backend_example.whom
 * This is the original example used in Backend.java.
 */

realtime number i;
number b=1;

class Light {
	boolean on;
	void switchOn() {
		on = true;
	}
}

Light light1;

once (i>0)
{
	b=b+1;
	light1.switchOn();
}
```

Toybox

```
/* arvid1.whom */
// the number that is observed
realtime number i=0;

// our standard test number
number b=0;

class foo {
    number c=1;
    number d=2;

    void bar() {
      c=d;
    }

    number baz() {
      return c+d;
      // will this get executed?
      b = 5;
    }
}

foo foo1;
foo foo2;

once(i) {
    foo1.bar();
```

```
    b = foo2.baz();
    b = b + foo1.baz();
}

/* backend1.whom */
realtime number i=0;
number b=0;

class Alarm
{
    sound;
    void bar()
      {
            sound = true;
    }
}

class Light
{
    Alarm alarm;
    number foo=1;

    EVENT_ON;
    EVENT_OFF;

    void switchOn()
      {
            dispatch EVENT_ON;
            //EVENT_ON = true;
    }

    void switchOff()
      {
            EVENT_ON = false;
    }

    number complicated( number one, number two )
      {
            // see which one is visible
            number two = 3;
            one = one + two;
            two = two * one;
            return two;
    }
}

Light light1;

once (i > 0)
{
    b=light1.complicated(b,i);
    light1.switchOn();
}

once light1.EVENT_ON
{
    b=b+1;
    light1.alarm.bar();
}

once light1.alarm.sound
{
    b = b+1;
    light1.switchOff();
    i=i-1;
```

```
}

/* fibonacci.whom */
// let's try to compute the Fibonacci numbers recursively

// the number that is observed
realtime number i=0;

// our standard test number
number b=0;

class foo {
    number fib(number x) {
      // this is ineffective by purpose ;-)
      if( x == 0 )
          return 0;
      else if( x == 1)
          return 1;
      else
          return fib(x-1) + fib(x-2);
    }
}

foo foo1;

once(i) {
    b = foo1.fib(i);
}

/* stat_sem_ana_err.whom */
/* This tests some static semantic analysis */


// the number that is observed
realtime number i=0;

number x;

string s1, s2;

class foo {
    number i;
    bar;

    string baz() {
      return "Hello World!";
    }

    number bla() {
      // error, should be reported by something other than a
ClassCastException
      return bar;
    }
}

foo f1;

once(i) {
    // this should generate an error
    s1 = s1 + s2;

    s1 = "bla";

    // error
    s2 = f1.bla();
```

```
    s2 = f1.baz();

    // no error here, but the return is not correct
    x = f1.bla();
}
```

Experimental

```
/*
 * polymorphism_test1.whom
 * Testing polymorphism.
 */

realtime number i;

class A {

    number foo() {
      return 3;
    }

    number bar() {
      return foo();
    }
}

class B extends A {
    number foo() {
      return 5;
      // this should not be reached
      i=i+1;
    }

    number baz=0;
}

B b;

once(i) {
    b.baz = b.bar();
}
```