```java
/*
 * Created on Jun 27, 2005
 *
 * TODO To change the template for this generated file go to
 * Window − Preferences − Java − Code Generation − Code and Comments
 */
package orc;
import java.io.FileInputStream;
import java.io.InputStream;

import orc.ast.OrcProcess;
import orc.parser.OrcLexer;
import orc.parser.OrcParser;
import orc.runtime.OrcEngine;
import orc.runtime.Token;
import orc.runtime.nodes.Node;

/**
 * Main class for Orc. Parses Orc file and executes it
 * @author wcook
 */
public class Orc {

    /**
     * Standard made program. Arguments are −debug are Orc file name (or s
tandard input).
     * @param args
     */
    public static void main(String[] args) {
        OrcEngine engine = new OrcEngine();
        try {
            int i = 0;
            if (args.length > i && args[i].equals("−debug")) {
                i++;
                engine.debugMode = true;
            }
            InputStream in;
            if (args.length == i)
                in = System.in;
            else
                in = new FileInputStream(args[i]);
            OrcLexer lexer = new OrcLexer(in);
            OrcParser parser = new OrcParser(lexer);
            OrcProcess p = parser.startRule();

            engine.run(p.compile(new PrintResult()));

        } catch (Exception e) {
            System.err.println("exception: " + e);
            if (engine.debugMode)
                e.printStackTrace();
        } catch (Error e) {
            System.err.println(e.toString());
            if (engine.debugMode)
                e.printStackTrace();
        }
    }
}

/**
 * A special node that prints its output.
 * Equivalent to
 * <pre>
 *    P >x> println(x)
```

```java
 * </pre>
 * @author wcook
 */
class PrintResult extends Node {
    public void process(Token t, OrcEngine engine) {
        Object val = t.getResult();
        System.out.println(val.toString());
        System.out.flush();
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime;

import java.util.LinkedList;

import orc.runtime.nodes.Node;
import orc.runtime.sites.Calc;
import orc.runtime.sites.Let;
import orc.runtime.sites.Mail;
import orc.runtime.sites.Rtimer;
import orc.runtime.values.Constant;
import orc.runtime.values.GroupCell;

/**
 * The Orc Engine provides the main look for executing active tokens.
 * @author wcook
 */
public class OrcEngine {

    LinkedList<Token> activeTokens = new LinkedList<Token>();
    LinkedList<Token> queuedReturns = new LinkedList<Token>();
    int calls;
    public boolean debugMode = false;

    /**
     * Run Orc given a root node.
     * Creates an initial environment and then
     * executes the main loop.
     * @param root  node to run
     */
    public void run(Node root) {

        GroupCell startGroup = new GroupCell();
        Token start = new Token(root, null/*env*/, null/* caller */, start
Group,
                null/* value */);

        start.bind("let", new Let());

        start.bind("cat", new Calc(Calc.Op.CAT));

        start.bind("add", new Calc(Calc.Op.ADD));
        start.bind("sub", new Calc(Calc.Op.SUB));
        start.bind("mul", new Calc(Calc.Op.MUL));
        start.bind("div", new Calc(Calc.Op.DIV));

        start.bind("lt", new Calc(Calc.Op.LT));
        start.bind("le", new Calc(Calc.Op.LE));
        start.bind("eq", new Calc(Calc.Op.EQ));
        start.bind("ne", new Calc(Calc.Op.NE));
        start.bind("ge", new Calc(Calc.Op.GE));
        start.bind("gt", new Calc(Calc.Op.GT));

        start.bind("and", new Calc(Calc.Op.AND));
        start.bind("or", new Calc(Calc.Op.OR));
        start.bind("not", new Calc(Calc.Op.NOT));

        start.bind("random", new Calc(Calc.Op.RAND));
        start.bind("if", new Calc(Calc.Op.IF));

        start.bind("item", new Calc(Calc.Op.ITEM));
        start.bind("print", new Calc(Calc.Op.PRINT));
```

```
        start.bind("println", new Calc(Calc.Op.PRINTLN));

        start.bind("Rtimer", new Rtimer());
        try {
            start.bind("SendMail", new Mail());
        } catch (Error e) {
            System.err.println("Warning: mail not avaiable (" + e + ")");
        }

        start.bind("true", new Constant(Boolean.TRUE));
        start.bind("false", new Constant(Boolean.FALSE));

        activeTokens.add(start);

        int round = 1;
        while (moreWork()) {
            if (debugMode)
                debug("** Round " + (round++) + " ***", null);

            while (activeTokens.size() > 0)
                activeTokens.remove().process(this);

            if (queuedReturns.size() > 0)
                activeTokens.add(queuedReturns.remove());
        }
    }

    /**
     * Internal function to check if there is more work to do
     * @return true if more work
     */
    private synchronized boolean moreWork() {
        if (activeTokens.size() == 0) {
            if (calls == 0)
                return false;
            try {
                wait();
            } catch (InterruptedException e) {
            }
        }
        return true;
    }

    /**
     * Activate a token by adding it to the queue of active tokens
     * @param t the token to be added
     */
    synchronized public void activate(Token t) {
        activeTokens.addLast(t);
        notify();
    }

    /**
     * Counts how many calls have been made
     * TODO: this is a hack only needed to identify when Orc
     * can terminate. Normally an Orc execution would terminate
     * when the first value is produced, and this count would
     * not be needed.
     * @param n
     */
    public void addCall(int n) {
        calls += n;
    }
    /**
     * Called when a site returns a value. Add the corresponding
     * token to queue of returned sites
     * @param label
     * @param token
     * @param value
     */
    synchronized public void siteReturn(String label, Token token,
            Object value) {
        token.setResult(value);
        queuedReturns.add(token);
        if (debugMode)
            debug("ASYMC: " + label + " returned: " + value, token);
        notify(); // wake up main thread
    }

    public void debug(String string, Token token) {
        // if (token != null)
        // System.out.print("[" + token.hashCode() + "] ");
        System.out.println(string);
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime;

import orc.runtime.nodes.Node;
import orc.runtime.values.GroupCell;
import orc.runtime.values.Value;

/**
 * Representation of an active thread of execution. Tokens
 * move over the node graph as they are executed. They contain
 * an environment, and may be low to a group. They also
 * preserve the call chain and contain a value to be passed
 * to the next token.
 * @author wcook
 */
public class Token {
    protected Node node;
    protected Environment env;
    protected GroupCell group;
    Token caller;
    Object result;

    public Token(Node node, Environment env, Token caller, GroupCell group
, Object result) {
        this.node = node;
        this.env = env;
        this.caller = caller;
        this.group = group;
        this.result = result;
    }

    /**
     * If a token is alive, calls the node to perform the next action
     * @param engine
     */
    public void process(OrcEngine engine) {
        if (group.isAlive())
            node.process(this, engine);
    }

    public Node getNode() {
        return node;
    }

    public GroupCell getGroup() {
        return group;
    }

    public Token setGroup(GroupCell group) {
        this.group = group;
        return this;
    }

    /**
     * Move to a node node
     * @param node  the node to move to
     * @return  returns self
     */
    public Token move(Node node) {
        this.node = node;
        return this;
    }
    /**
     * Create a copy of the token
     * @return  new token
     */
    public Token copy() {
        return new Token(node, env, caller, group, result);
    }

    /**
     * Extend the environment with a new variable/value pair
     * @param var   variable name
     * @param val   value for this variable
     * @return      self
     */
    public Token bind(String var, Value val) {
        env = new Environment(var, val, env);
        return this;
    }

    /**
     * Lookup a variable in the environment
     * @param var   variable name
     * @return      value, or an exception if the variable is undefined
     */
    public Value lookup(String var) {
        return env.lookup(var);
    }

    public Environment getEnvironment() {
        return env;
    }

    public Object getResult() {
        return result;
    }

    public Token setResult(Object result) {
        this.result = result;
        return this;
    }

    public Token getCaller() {
        return caller;
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime;

import orc.runtime.values.Value;

/**
 * Lexical environment containing variable bindings
 * @author wcook
 */
public class Environment {
    Environment parent;
    String var;
    Value value;

    public Environment(String var, Value value, Environment parent) {
        this.var = var;
        this.value = value;
        this.parent = parent;
    }

    /**
     * Lookup a variable in the environment
     * TODO: should be compiled using activation frames and variable offse
ts.
     * Currently uses a linear search.
     * @param var   variable name
     * @return      value, or error if binding exists
     */
    public Value lookup(String var) {
        if (this.var.equals(var))
            return value;
        else if (parent == null)
            throw new Error("Undefined variable: " + var);
        else
            return parent.lookup(var);
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.nodes;

import orc.runtime.OrcEngine;
import orc.runtime.Token;

/**
 * Abstract base class for compile nodes
 * @author wcook
 */
public abstract class Node {
    /**
     * The process method is the fundamental opreation in the execution en
gine.
     * It is called to perform the action of the node given a token and
     * the execution engine.
     * @param t      input token being processed
     * @param engine used to activate the next token
     */
    public abstract void process(Token t, OrcEngine engine);
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.nodes;

import orc.runtime.OrcEngine;
import orc.runtime.Token;

/**
 * A compile node that performs a fork to run two subnodes.
 * @author wcook
 */
public class Fork extends Node {
    Node left;
    Node right;
    public Fork(Node left, Node right) {
        this.left = left;
        this.right = right;
    }

    /**
     * The input token is activated on the right node,
     * and a copy is activated on the left node.
     * @see orc.runtime.nodes.Node#process(orc.runtime.Token, orc.runtime.
OrcEngine)
     */
    public void process(Token t, OrcEngine engine) {
        if (engine.debugMode)
            engine.debug("Fork", t);

        engine.activate(t.copy().move(left));
        engine.activate(t.move(right));
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.nodes;

import orc.runtime.OrcEngine;
import orc.runtime.Token;
import orc.runtime.values.Constant;

/**
 * Compiled node for assignment.
 * @author wcook
 */
public class Assign extends Node {
    String var;
    Node next;

    public Assign(String var, Node next) {
        this.var = var;
        this.next = next;
    }

    /**
     * When executed, extends the environment with a new binding.
     * The result value in the input token is bound to the variable name.
     * The next node is activated.
     * @see orc.runtime.nodes.Node#process(orc.runtime.Token, orc.runtime.
OrcEngine)
     */
    public void process(Token t, OrcEngine engine) {
        if (engine.debugMode)
            engine.debug("Assign " + var + "=" + t.getResult(), t);

        Object val = t.getResult();
        t.bind(var, new Constant(val));
        engine.activate(t.move(next));
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.nodes;

import orc.runtime.OrcEngine;
import orc.runtime.Token;
import orc.runtime.values.GroupCell;

/**
 * A compiled where node
 * @author wcook
 */
public class Where extends Node {
    Node left;
    String var;
    Node right;

    public Where(Node left, String var, Node right) {
        this.left = left;
        this.var = var;
        this.right = right;
    }

    /**
     * Executing a where node creates a new group within the current group
.
     * The input token is copied and the variable is
     * associated with this group cell for execution of the
     * left side of the where. The token is then moved to the
     * right side and it is associated with the new group.
     * TODO: this could be expressed slightly better by adding a create gr
oup
     * call to a token.
     * @see orc.runtime.nodes.Node#process(orc.runtime.Token, orc.runtime.
OrcEngine)
     */
    public void process(Token t, OrcEngine engine) {
        if (engine.debugMode)
            engine.debug("Where " + var, t);

        GroupCell cell = t.getGroup().createCell();
        engine.activate(t.copy().bind(var, cell).move(left));
        engine.activate(t.move(right).setGroup(cell));
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.nodes;

import orc.runtime.OrcEngine;
import orc.runtime.Token;
import orc.runtime.values.GroupCell;

/**
 * Compiled node used to store the value of a binding in a where clause.
 * @author wcook
 */
public class Store extends Node {
    String var;
    public Store(String var) {
        this.var = var;
    }

    /**
     * Gets the group of the token and sets its value to be the result
     * of the input token.
     * As a side effect of setting the value of a group, a "where" variabl
e
     * becomes bound and the execution of the group is suspended.
     * @see orc.runtime.nodes.Node#process(orc.runtime.Token, orc.runtime.
OrcEngine)
     */
    public void process(Token t, OrcEngine engine) {
        if (engine.debugMode)
            engine.debug("Store/Stop " + var + "=" + t.getResult(), t);

        GroupCell group = t.getGroup();
        Object result = t.getResult();
        group.setValue(result, engine);
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.nodes;

import java.util.List;

import orc.runtime.OrcEngine;
import orc.runtime.Token;
import orc.runtime.values.Closure;

/**
 * Compiled node to create a definition
 * @author wcook
 */
public class Define extends Node {

    String name;
    List<String> formals;
    Node body;
    Node next;

    public Define(String name, List<String> formals, Node body, Node next)
 {
        this.name = name;
        this.formals = formals;
        this.body = body;
        this.next = next;
    }

    /**
     * Creates a closure containing the body of the definition. The enviro
nment
     * for the closure is the same as the input environment, but it is ext
ended
     * to <it>include a binding for the definition name whose value is the
 closure</it>.
     * This means that the closure environment must refer to the closure,
so there
     * is a cycle in the object pointer graph. This cycle is constructed i
n
     * three steps:
     * <nl>
     * <li>Create the closure with a null environment
     * <li>Bind the name to the new closure
     * <li>Update the closure to point to the new environment
     * </ul>
     * Then the next token is activated in this new environment.
     * This is a standard technique for creating recursive closures.
     * @see orc.runtime.nodes.Node#process(orc.runtime.Token, orc.runtime.
OrcEngine)
     */
    public void process(Token t, OrcEngine engine) {
        if (engine.debugMode)
            engine.debug("Define " + name, t);

        // create a recursive closure
        Closure c = new Closure(formals, body, null/*empty environment*/);
        t.bind(name, c);
        c.setEnvironment(t.getEnvironment());
        engine.activate(t.move(next));
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.nodes;

import java.util.List;

import orc.runtime.OrcEngine;
import orc.runtime.Token;
import orc.runtime.values.Callable;
import orc.runtime.values.Value;

/**
 * Compiled node for a call (either a site call or a definition call)
 * @author wcook
 */
public class Call extends Node {

    String name;
    List<Param> args;
    Node next;

    public Call(String name, List<Param> args, Node next) {
        this.name = name;
        this.args = args;
        this.next = next;
    }

    /**
     * Looks up the function to be called, then creates a call
     * token using the argument expressions.
     * TODO: why does this check for callable?
     * @see orc.runtime.nodes.Node#process(orc.runtime.Token, orc.runtime.
OrcEngine)
     */
    public void process(Token t, OrcEngine engine) {

        Value d = t.lookup(name);
        // define call with return location
        if (d instanceof Callable) {
            Callable target = (Callable) d;
            target.createCall(name, t, args, next, engine);
        }
        else
            t.setResult(d);
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.nodes;

import orc.runtime.OrcEngine;
import orc.runtime.Token;

/**
 * Compiled node marking the end of a procedure
 * @author wcook
 */
public class Return extends Node {

    /**
     * To execute a return, the caller token and the result of the current
     * execution are identified.
     * The caller token points to the node after the call.
     * The caller is then copied, the result of the caller is set, and
     * the token is activated.
     * @see orc.runtime.nodes.Node#process(orc.runtime.Token, orc.runtime.
OrcEngine)
     */
    public void process(Token t, OrcEngine engine) {
        if (engine.debugMode)
            engine.debug("Return " + t.getResult(), t);

        Token caller = t.getCaller();
        Object result = t.getResult();
        engine.activate(caller.copy().setResult(result));
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.nodes;

import orc.runtime.Token;
import orc.runtime.values.Value;

/**
 * Interface for parameters to calls.
 * @author wcook
 */
public interface Param {

    /**
     * Determine if the parameter is unbound in an environment
     * @param env   the environment containing bindings
     * @return      true if the parameter is unbound
     */
    boolean waitOnUnboundVar(Token env);

    /**
     * Gets the value of a parameter in an environment
     * @param env   the environment containing bindings
     * @return      the parameter value
     */
    Value getValue(Token env);
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.nodes;

import orc.runtime.OrcEngine;
import orc.runtime.Token;
import orc.runtime.values.Constant;
import orc.runtime.values.Value;

/**
 * A compiled literal node
 * @author wcook
 */
public class Literal extends Node implements Param {

    Object value;
    Node next;

    public Literal(Object value, Node next) {
        this.value = value;
        this.next = next;
    }

    /**
     * Executing a literal sets the value of the token and then activates
the next node.
     * @see orc.runtime.nodes.Node#process(orc.runtime.Token, orc.runtime.
OrcEngine)
     */
    public void process(Token t, OrcEngine engine) {
        t.setResult(value);
        engine.activate( t.move(next) );
    }

    /**
     * Creates a constant container for the literal value
     * @see orc.runtime.nodes.Param#getValue(orc.runtime.Token)
     */
    public Value getValue(Token env) {
        return new Constant(value);
    }

    /**
     * Literals are never unbound.
     * @see orc.runtime.nodes.Param#waitOnUnboundVar(orc.runtime.Token)
     */
    public boolean waitOnUnboundVar(Token env) {
        return false;
    }

    public String toString() {
        if (value instanceof String)
            return "\"" + value + "\"";
        else
            return value.toString();
    }
}
```

```java
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.nodes;

import orc.runtime.Token;
import orc.runtime.values.GroupCell;
import orc.runtime.values.Value;

/**
 * A compiled variable node
 * @author wcook
 */
public class Variable implements Param {
    String var;

    public Variable(String var) {
        this.var = var;
    }

    /**
     * Looks up the variable to see if it is bound.
     * If the variable is bound to a constant, then it will
     * never be unbound. If the variable is associated with a group,
     * then it may be unbound.
     * If the group is unbound, then the input token is added to the
     * waiting queue for the group.
     * @see orc.runtime.nodes.Param#waitOnUnboundVar(orc.runtime.Token)
     */
    public boolean waitOnUnboundVar(Token t) {
        Value holder = t.lookup(var);
        GroupCell cell = holder.asUnboundCell();
        if (cell == null)
            return false;
        cell.waitForValue(t);
        return true;
    }

    /**
     * Looks up the value in of the variable in the environment.
     * @see orc.runtime.nodes.Param#getValue(orc.runtime.Token)
     */
    public Value getValue(Token env) {
        return env.lookup(var);
    }

    public String toString() {
        return var;
    }
}
```

```java
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.values;

/**
 * Interface for value containers
 * @author wcook
 */
public interface Value {

    /**
     * Check if a value container is bound
     * @return true if it is unbound
     */
    GroupCell asUnboundCell();

    /**
     * If the container is bound, return the underlying java value
     * @return any value
     */
    Object asBasicValue();
}
```

```java
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.values;

/**
 * Base class that for value containers
 * @author wcook
 */
public class BaseValue implements Value {

    /**
     * Determine if the value is unbound
     *
     * @see orc.runtime.values.Value#asUnboundCell()
     */
    public GroupCell asUnboundCell() {
        return null;
    }

    /**
     * Extract the underlying Java value of the container
     *
     * @see orc.runtime.values.Value#asBasicValue()
     */
    public Object asBasicValue() {
        return this;
    }

}
```

```java
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.values;

/**
 * A value container for a literal value
 * @author wcook
 */
public class Constant extends BaseValue {

    Object value;

    public Constant(Object value) {
        this.value = value;
    }

    public Object asBasicValue() {
        return value;
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.values;

import java.util.Arrays;
import java.util.List;

/**
 * A tuple value container
 * @author wcook
 */
public class Tuple extends BaseValue {

    Object[] values;

    public Tuple(Object[] values) {
        this.values = values;
    }

    public Object at(int i) {
        return values[i];
    }

    public String toString() {
        return format('[', Arrays.asList(values), ",", ']');
    }

    public static String format(char left, List items, String sep, char ri
ght) {
        StringBuffer buf = new StringBuffer();
        buf.append(left);
        int i = 0;
        for (Object x : items) {
            if (i > 0)
                buf.append(sep);
            buf.append(x);
            i++;
        }
        buf.append(right);
        return buf.toString();
    }

    public int size() {
        return values.length;
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.values;

import java.util.ArrayList;
import java.util.List;

import orc.runtime.OrcEngine;
import orc.runtime.Token;

/**
 * A value container that is also a group. Groups are
 * essential to the evaluation of where clauses: all the
 * tokens that arise from execution of a where definition
 * are associated with the same group. Once a value is
 * produced for the group, all these tokens are terminated.
 * @author wcook
 */
public class GroupCell implements Value {

    Object value;
    boolean bound;
    boolean alive;
    List<Token> waitList;
    List<GroupCell> children;

    public GroupCell() {
        bound = false;
        alive = true;
    }

    /**
     * A group is unbound as long as no value has been produced
     * @see orc.runtime.values.Value#asUnboundCell()
     */
    public GroupCell asUnboundCell() {
        return bound ? null : this;
    }

    /**
     * Once the group is bound, its value can be accessed.
     * @see orc.runtime.values.Value#asBasicValue()
     */
    public Object asBasicValue() {
        if (!bound)
            throw new Error("Getting value of unbound cell");
        return value;
    }

    /**
     * Groups are organized into a tree. In this case a new
     * subgroup is created and returned
     * @return the new group
     */
    public GroupCell createCell() {
        GroupCell n = new GroupCell();
        if (children == null)
            children = new ArrayList<GroupCell>();
        children.add(n);
        return n;
    }

    /**
```

```java
     * This call defines the fundamental behavior of groups:
     * When the value is bound, all subgroups are killed
     * and all waiting tokens are activated.
     * @param value     the value for the group
     * @param engine    engine
     */
    public void setValue(Object value, OrcEngine engine) {
        this.value = value;
        bound = true;
        kill();
        if (waitList != null)
            for (Token t : waitList)
                engine.activate(t);
    }

    /**
     * Recursively kills all subgroups
     */
    private void kill() {
        alive = false;
        if (children != null)
            for (GroupCell sub : children)
                sub.kill();
    }

    /**
     * Check if a group has been killed
     * @return true if the group has not been killed
     */
    public boolean isAlive() {
        return alive;
    }

    /**
     * Add a token to the waiting queue of this group
     * @param t
     */
    public void waitForValue(Token t) {
        if (waitList == null)
            waitList = new ArrayList<Token>();
        waitList.add(t);
    }

}
```

```java
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.values;

import java.util.List;

import orc.runtime.OrcEngine;
import orc.runtime.Token;
import orc.runtime.nodes.Node;
import orc.runtime.nodes.Param;

/**
 * Callable objects include sites and definitions
 * @author wcook
 */
public interface Callable {

    /**
     * Create a call to a callable value
     * @param label     name (used for debugging)
     * @param caller    token for which the call is being made: points to
the call node
     * @param args      argumetn list
     * @param nextNode  next node after the call node, to which the result
 should be sent
     * @param engine    Orc engine
     */
    void createCall(String label, Token caller, List<Param> args,
            Node nextNode, OrcEngine engine);

}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.values;

import java.util.List;

import orc.runtime.Environment;
import orc.runtime.OrcEngine;
import orc.runtime.Token;
import orc.runtime.nodes.Node;
import orc.runtime.nodes.Param;
import orc.runtime.nodes.Return;

/**
 * Represents a standard closure: a function defined in an environment
 *
 * @author wcook
 */
public class Closure extends BaseValue implements Callable {

    Node body;
    List<String> formals;
    Environment env;

    public Closure(List<String> formals, Node body, Environment env) {
        this.body = body;
        this.formals = formals;
        this.env = env;
    }

    /**
     * To create a class to a closure, a new token is created using the
     * environment in which the closure was defined. This environment is
     * then extended to bind the formals to the actual arguments.
     * The caller of the new token is normally a token point to right
     * after the call. However, for tail−calls the existing caller
     * is reused, rather than creating a new intermediate stack frame.
     * @see orc.runtime.values.Callable#createCall(java.lang.String, orc.r
untime.Token, java.util.List, orc.runtime.nodes.Node, orc.runtime.OrcEngin
e)
     */
    public void createCall(String label, Token callToken,
            List<Param> args, Node nextNode, OrcEngine engine) {
        if (engine.debugMode)
            engine.debug("Call " + label + Tuple.format(’(’, args, ",", ’)’
),
                    callToken);

        GroupCell callGroup = callToken.getGroup();

        // check tail−call optimization
        Token returnToken;
        if (nextNode instanceof Return)
            returnToken = callToken.getCaller(); // tail−call
        else
            returnToken = callToken.move(nextNode); // normal call

        Token n = new Token(body, env, returnToken, callToken.getGroup(),
null/*value*/);

        int i = 0;
        for (Param e : args)
            n.bind(formals.get(i++), e.getValue(callToken));
```

```
        engine.activate(n);
    }

    public void setEnvironment(Environment env) {
        this.env = env;
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.sites;

import java.util.List;

import orc.runtime.OrcEngine;
import orc.runtime.Token;
import orc.runtime.nodes.Node;
import orc.runtime.nodes.Param;
import orc.runtime.values.BaseValue;
import orc.runtime.values.Callable;
import orc.runtime.values.Tuple;

/**
 * Base class for all sites
 * @author wcook
 */
public abstract class Site extends BaseValue implements Callable {

    /**
     * Invoked by a Call to invoke a site. The argument list is
     * scanned to make sure that all parameters are bound.
     * If an unbound parameter is found, the call is placed on a
     * queue and nothing more is done.
     * Once all parameters are bound, their values are collected
     * and the corresponding subclass (the actual site) is called.
     *
     * @see orc.runtime.values.Callable#createCall(java.lang.String, orc.r
untime.Token, java.util.List, orc.runtime.nodes.Node, orc.runtime.OrcEngin
e)
     */
    public void createCall(String label, Token callToken,
            List<Param> args, Node nextNode, OrcEngine engine) {

        for (Param e : args)
            if (e.waitOnUnboundVar(callToken)) {
                if (engine.debugMode)
                    engine.debug("Wait " + label + " for " + e, callToken);
                return;
            }

        Object[] values = new Object[args.size()];
        int i = 0;
        for (Param e : args)
            values[i++] = e.getValue(callToken).asBasicValue();

        if (engine.debugMode)
            engine.debug("Call site " + label + new Tuple(values), callToken);

        callSite(values, callToken.move(nextNode), engine);
    }

    /**
     * Must be implemented by subclasses to implement the site behavior
     * @param args          list of argument values
     * @param returnToken   where the result should be sent
     * @param engine        Orc engine -- used for suspending or activatin
g tokens
     */
    abstract void callSite(Object[] args, Token returnToken,
            OrcEngine engine);
```

```
    /**
     * Helper function for integers
     */
    int intArg(Object[] args, int n) {
        return ((Integer)args[n]).intValue();
    }

    /**
     * Helper function for booleans
     */
    boolean boolArg(Object[] args, int n) {
        return ((Boolean)args[n]).booleanValue();
    }

    /**
     * Helper function for strings
     */
    String stringArg(Object[] args, int n) {
        return args[n].toString();
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.sites;

import orc.runtime.OrcEngine;
import orc.runtime.Token;
import orc.runtime.values.Tuple;

/**
 * Implements the built−in "let" site
 * @author wcook
 */
public class Let extends Site {

    /**
     *  Outputs a single value or creates a tuple.
     * @see orc.runtime.sites.Site#callSite(java.lang.Object[], orc.runtim
e.Token, orc.runtime.OrcEngine)
     */
    void callSite(Object[] args, Token returnToken, OrcEngine engine) {

        Object value = (args.length == 1) ? args[0] : new Tuple(args);
        returnToken.setResult(value);
        engine.activate(returnToken);
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.sites;

import java.util.Random;

import orc.runtime.OrcEngine;
import orc.runtime.Token;
import orc.runtime.values.Tuple;

/**
 * Helper class defining many basic sites.
 * @author wcook
 */
public class Calc extends Site {

    public enum Op {
        ADD, SUB, MUL, DIV,
        LT, LE, EQ, NE, GE, GT,
        AND, OR, NOT,
        IF,
        RAND,
        ITEM,
        CAT,
        PRINT, PRINTLN
        } ;
    Op op;

    Random random = new Random();

    public Calc(Op op) {
        this.op = op;
    }

    /**
     * Performs the computation for all the basic calculation sites.
     * @see orc.runtime.sites.Site#callSite(java.lang.Object[], orc.runtim
e.Token, orc.runtime.OrcEngine)
     */
    void callSite(Object[] args, Token returnToken, OrcEngine engine) {

        int n = 2;
        Object result = null;
        switch (op) {
            case CAT: {
                StringBuffer buf = new StringBuffer();
                for (Object x : args)
                    buf.append(x.toString());
                result = buf.toString();
                n = args.length;
                break;
            }

            case ADD: result = Integer.valueOf(intArg(args, 0) + intArg(ar
gs, 1)); break;
            case SUB: result = Integer.valueOf(intArg(args, 0) − intArg(ar
gs, 1)); break;
            case MUL: result = Integer.valueOf(intArg(args, 0) * intArg(ar
gs, 1)); break;
            case DIV: result = Integer.valueOf(intArg(args, 0) / intArg(ar
gs, 1)); break;

            case LE: result = Boolean.valueOf(intArg(args, 0) <= intArg(ar
gs, 1)); break;
            case LT: result = Boolean.valueOf(intArg(args, 0) < intArg(arg
s, 1)); break;
            case EQ: result = Boolean.valueOf(intArg(args, 0) == intArg(ar
gs, 1)); break;
            case NE: result = Boolean.valueOf(intArg(args, 0) != intArg(ar
gs, 1)); break;
            case GT: result = Boolean.valueOf(intArg(args, 0) > intArg(arg
s, 1)); break;
            case GE: result = Boolean.valueOf(intArg(args, 0) >= intArg(ar
gs, 1)); break;

            case RAND: result = Integer.valueOf(random.nextInt(intArg(args
, 0))); n = 1; break;

            case AND: result = Boolean.valueOf(boolArg(args, 0) && boolArg
(args, 1)); break;
            case OR: result = Boolean.valueOf(boolArg(args, 0) || boolArg(
args, 1)); break;
            case NOT: result = Boolean.valueOf(!boolArg(args, 0)); n = 1;
break;

            case IF: {
                if (boolArg(args, 0))
                    result = true;
                n = 1;
                break;
            }

            case ITEM: {
                Object v = args[0];
                int m = intArg(args, 1);
                n = 2;
                if (v instanceof String)
                    if (args.length == 3)
                    {
                        n = 3;
                        result = ((String)v).substring(m, intArg(args, 2))
;
                    }
                    else
                        result = ((String)v).substring(m, m + 1);
                else if (v instanceof Tuple)
                    result = ((Tuple)v).at(m);
                else
                    throw new Error("Invalid item access");
                break;
            }

            case PRINTLN:
            case PRINT: {
                for (Object x : args)
                    System.out.print(x.toString());
                if (op == Op.PRINTLN)
                    System.out.println();
                n = args.length;
                result = true;
                break;
            }
        }

        if (args.length != n)
            throw new Error("Expected " + n + " arguments for " + op + args);
```

```
        if (result != null) {
            returnToken.setResult(result);
            engine.activate(returnToken);
        }
    }

}
```

```java
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.sites;

import java.util.PriorityQueue;

import orc.runtime.OrcEngine;
import orc.runtime.Token;
import orc.runtime.values.Tuple;

/**
 * Implements the RTimer site
 *  * @author wcook
 */
public class Rtimer extends Site {

    /**
     * When called, the RTimer creates a new thread which wakes up after s
ome time and returns the value
     * @see orc.runtime.sites.Site#callSite(java.lang.Object[], orc.runtim
e.Token, orc.runtime.OrcEngine)
     *
     */
    public static JavaTimer javaTimer;

    public void callSite(Object[] args, Token returnToken, OrcEngine engin
e)
    {
        if (args.length != 1 || !(args[0] instanceof Integer))
            throw new Error("Invalid argument in Rtimer" + args);
        engine.addCall(1);

        long n = ((Integer) args[0]).longValue();
        if (javaTimer == null)
            javaTimer = new JavaTimer(engine);
        javaTimer.addEvent(n, returnToken);
    }
}


/**
 * Helper class that runs the actual timer and calls Rtimer Events
 * @author jayeshs
 */

class JavaTimer implements Runnable {

    /**
     * JavaTimer is instantiated as a static object javaTimer in  orc.runt
ime.sites.Rtimer.
     * It creates a PriorityQueue object , which is used to store events s
cheduled in relative time
     * by calls to Rtimer, and spawns a thread to remove events from the q
ueue and return an engine call.
     */

    Thread t;
    PriorityQueue<RtimerQueueEntry> rtimerEventQueue;
    OrcEngine engine;

    public JavaTimer(OrcEngine engine)
    {
        this.engine = engine;
```

```java
        rtimerEventQueue = new PriorityQueue<RtimerQueueEntry>();
    }

    public void addEvent(long time, Token token)
    {
        if (t == null) {
            t = new Thread(this);
            t.start();
            if (engine.debugMode)
                engine.debug("Rtimer: Starting Timer Thread.",token);

        }
        long at = time + System.currentTimeMillis();
        rtimerEventQueue.add(new RtimerQueueEntry(at, token));
        if (engine.debugMode)
            engine.debug("Rtimer: Adding event to Rtimer Event Queue.",token);
        t.interrupt();

    }

    public synchronized void run() {
        while (true) {
            try
            {
                RtimerQueueEntry temp = rtimerEventQueue.peek();
                if (temp == null)
                {
                    // wait until interrupted
                    wait();
                }
                else if (temp.getTime() > System.currentTimeMillis())
                {
                    // wait for first event
                    if (engine.debugMode)
                        engine.debug("Rtimer: Waiting for " + (temp.getTime() −
System.currentTimeMillis()),temp.getToken());
                    wait(temp.getTime() − System.currentTimeMillis());
                }
                else
                {
                    // execute the event
                    rtimerEventQueue.remove();
                    engine.addCall(−1);
                    if (engine.debugMode)
                        engine.debug("Rtimer: Executed Event.",temp.getToken());

                    engine.siteReturn("Rtimer", temp.getToken(), true);

                }
            }
            catch(InterruptedException e)
            {
                /*something added to queue */

            }

        }
    }



    /**
     * Class representing Rtimer Queue Entry
     * @author jayeshs
     */
    class RtimerQueueEntry implements Comparable<RtimerQueueEntry> {

        long time;
        Token token;

        public RtimerQueueEntry(long time, Token token) {
            this.token = token;
            this.time = time;
        }
        public long getTime() {
            return time;
        }
        public Token getToken() {
            return token;
        }

        // sort the queue items earliest first
        public int compareTo(RtimerQueueEntry n) {
            long diff = time − n.time;

            if (diff == 0)
                return 0;
            else if (diff > 0)
                return 1;
            else
                return −1;
        }
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.runtime.sites;

import java.util.Properties;

import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

import orc.runtime.OrcEngine;
import orc.runtime.Token;
import orc.runtime.values.Tuple;

/**
 * Implements mail sending
 * @author wcook
 */
public class Mail extends Site {

    /**
     * Uses the java mail API to send a message via an SMTP server
     * TODO: there are many possible enhancements of this code
     * @see orc.runtime.sites.Site#callSite(java.lang.Object[], orc.runtim
e.Token, orc.runtime.OrcEngine)
     */
    void callSite(Object[] args, Token returnToken, OrcEngine engine) {
        if (args.length != 5)
            throw new Error("sendEmail(from, to, subject, message, smtp)");

        String from = stringArg(args, 0);
        Tuple to;
        if (args[1] instanceof Tuple)
            to = (Tuple) args[1];
        else
            to = new Tuple(new Object[]{stringArg(args, 1)});
        String subject = stringArg(args, 2);
        String message = stringArg(args, 3);
        String smtp = stringArg(args, 4);

        // Set the host smtp address
        Properties props = new Properties();
        props.setProperty("mail.smtp.host", smtp);

        // create some properties and get the default Session
        Session session = Session.getDefaultInstance(props, null);
        // session.setDebug(debug);

        // create a message
        Message msg = new MimeMessage(session);

        // set the from and to address
        InternetAddress addressFrom;
        try {
            addressFrom = new InternetAddress(from);
            msg.setFrom(addressFrom);

            InternetAddress[] addressTo = new InternetAddress[to.size()];
            for (int i = 0; i < to.size(); i++) {
                addressTo[i] = new InternetAddress(to.at(i).toString());
            }
            msg.setRecipients(Message.RecipientType.TO, addressTo);

            // Optional : You can also set your custom headers in the Emai
l
            // msg.addHeader("MyHeaderName", "myHeaderValue");

            // Setting the Subject and Content Type
            msg.setSubject(subject);
            msg.setContent(message, "text/plain");
            Transport.send(msg);
        } catch (Exception e) {
            throw new Error(e.toString());
        }
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */

header {
    package orc.parser;

    import java.util.*;
    import java.io.FileInputStream;
    import java.io.FileNotFoundException;
    import orc.ast.*;
}

class OrcParser extends Parser;

startRule returns [OrcProcess n = null]
    : n=expr
    ;

expr returns [OrcProcess n = null]
    : n=def
    | n=where_expr
    | n=import_expr
    ;

import_expr returns [OrcProcess n=null]
    {
        OrcProcess m;
    }
    : "import" sl:STRING m=expr
        {
         OrcLexer lexer=null;
         try {
                        lexer = new OrcLexer(new FileInputStream(sl.getTex
t()));
                    } catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
         OrcParser parser = new OrcParser(lexer);
         n = new Include(parser.startRule(), m);
        }
    ;


def returns [OrcProcess n = null]
    {
        OrcProcess body, rest;
        List<String> formals;
    }
    : "def" name:NAME formals=formals_list EQ body=expr rest=expr
        { n = new Define(name.getText(), formals, body, rest); }
    ;

formals_list returns [List<String> formals = new ArrayList<String>() ]
    : ( LPAREN n:NAME
            { formals.add(n.getText()); }
        ( COMMA n2:NAME
            { formals.add(n2.getText()); }
        )* RPAREN
      )?
    ;

where_expr returns [OrcProcess n = null]
```

```
    : n=par_expr (
        "where"
            { AsymmetricParallelComposition
                an = new AsymmetricParallelComposition(n);
                n = an; }
            binding_list[an]
        )?
    ;

binding_list[AsymmetricParallelComposition n]
    : binding[n] ( SEMI binding[n] )*
    ;

binding[AsymmetricParallelComposition n]
    {
        OrcProcess expr;
    }
    : name:NAME "in" expr=par_expr
        { n.addBinding(name.getText(), expr); }
    ;

par_expr returns [OrcProcess n = null]
    {
        OrcProcess n2;
    }
    : n=seq_expr (
        PAR n2=seq_expr
            { n = new ParallelComposition(n, n2); }
        )*
    ;

seq_expr returns [OrcProcess n = null]
    {
        OrcProcess n2;
    }
    : n=basic_expr[false] (
        var:SEQ n2=seq_expr
            { n = new SequentialComposition(n, var.getText(), false, n2);
}
        | var2:SEQPUB n2=seq_expr
            { n = new SequentialComposition(n, var2.getText(), true, n2);
}
        )?
    ;

basic_expr[boolean asParam] returns [OrcProcess n = null]
    {
        List<OrcProcess> args = null;
        OrcProcess p;
    }
    : LBRACE n=expr RBRACE
    | name:NAME ( LPAREN
                { args = new ArrayList<OrcProcess>(); }
            p=basic_expr[true]
                { args.add(p); }
            ( COMMA p=basic_expr[true] { args.add(p); })*
            RPAREN )?
        { if (asParam && args == null)
            n = new Variable(name.getText());
          else
             n = new Call(name.getText(), args); }
    | num:INT
        { n = new Literal(new Integer(num.getText())); }
    | str:STRING
```

```
        { n = new Literal(str.getText()); }
    ;


class OrcLexer extends Lexer;

options {
    charVocabulary = '\3'..'\177';
    k = 2;
}

SL_COMMENT:
    "--" (~'\n')* '\n'
     { newline(); $setType(Token.SKIP); }
     ;

protected
BEGIN_COMMENT: LBRACE '-';
protected
END_COMMENT: '-' RBRACE;

/*
ML_COMMENT:
    LBRACE '-' (options {
        generateAmbigWarnings=false;
      }:  { LA(2)!=RBRACE }? '-'
      | '\n' {newline();}
      | ~('-'|'\n')
    )*
    '-' RBRACE
    {$setType(Token.SKIP);}
;

ML_COMMENT:
    BEGIN_COMMENT ( options {
        generateAmbigWarnings=false;
      }:{LA(2) != '}'}? '-'
    | '\n' {newline();}
    | ~('-'|'\n') )*
    END_COMMENT
    {$setType(Token.SKIP);}
;
*/


ML_COMMENT:
    BEGIN_COMMENT ( options {greedy=false;} :'\n' {newline();} | ~'\n')* E
ND_COMMENT
    {$setType(Token.SKIP);}
;


// one-or-more letters followed by a newline
NAME :   ALPHA ( ALPHA | DIGIT )*
    ;

INT : ( DIGIT )+;

STRING: '"'! ( ESCAPE | ~('"'|'\\') )* '"'!;

protected
ESCAPE
  :  '\\'
     ( 'n' { $setText("\n"); }
    | 'r' { $setText("\r"); }
    | 't' { $setText("\t"); }
    |'"' { $setText("\""); }
         | '\\' { $setText("\\"); }
         )
    ;

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

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

protected
SEQPUB : ">!"! ( NAME )? ">"!  ;

protected
SEQ : ">"! ( NAME )? ">"!  ;

protected
LBRACE : '{';

protected
RBRACE : '}';

SEQ_OR_PUB :
    (SEQ) => SEQ { $setType(SEQ); }
    | (SEQPUB) => SEQPUB { $setType(SEQPUB); }
    ;

PAR : '|';
SEMI: ';';
COMMA: ',';
EQ: '=';
LPAREN : '(';
RPAREN : ')';


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

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.ast;

import orc.runtime.nodes.Node;
import orc.runtime.nodes.Param;

/**
 * Base class for the abstract syntax tree
 * @author wcook
 */
abstract public class OrcProcess {

    /**
     * Compiles abstrac syntax tree into execution nodes.
     * Every node is compile relative to an "output" node that represents
     * the "rest of the program". Thus the tree of compiled nodes is creat
ed bottom up.
     * @param output IMPORTANT: this is the node to which output will be d
irected
     * @return A new node
     */
    public abstract Node compile(Node output);

    public Param asParam() {
        return null; // overriden by parameter types
    }
}
```

```java
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.ast;

import orc.runtime.nodes.Fork;
import orc.runtime.nodes.Node;

/**
 * Parallel compisition: left | right
 * @author wcook
 */
public class ParallelComposition extends OrcProcess {

    OrcProcess left;
    OrcProcess right;

    public ParallelComposition(OrcProcess left, OrcProcess right) {
        this.left = left;
        this.right = right;
    }

    /**
     * Creates a Fork node to run both left and right, which
     * both output to the same node.
     * @see orc.ast.OrcProcess#compile(orc.runtime.nodes.Node)
     */
    public Node compile(Node output) {
        return new Fork(left.compile(output), right.compile(output));
    }

    public String toString() {
        return "{" + left + "\n| " + right + "}";
    }
}
```

```java
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.ast;

import orc.runtime.nodes.Assign;
import orc.runtime.nodes.Fork;
import orc.runtime.nodes.Node;

/**
 * Abstract syntax tree for
 * <pre>
 *    left >[!] [var]> right
 * </pre>
 * Both ! and var are optional.
 * If ! is present, then publish is true and the var should be output.
 * @author wcook
 */
public class SequentialComposition extends OrcProcess {
    OrcProcess left;
    String var;
    boolean publish;
    OrcProcess right;

    public SequentialComposition(OrcProcess left, String var,
            boolean publish, OrcProcess right) {
        this.left = left;
        this.var = var;
        this.publish = publish;
        this.right = right;
    }

    /**
     * Compile the right side relative to the overall program output.
     * If the variable is present then create an
     * assign node.
     * If the result should be published, create a fork.
     * This is because
     * <pre>
     *    f >!v> g
     * </pre>
     * is equivalent to
     * <pre>
     *    f >v> (let(x) | g)
     * </pre>
     * Finally, compile the left side and send its output
     * to the newly created node for the right side.
     * @see orc.ast.OrcProcess#compile(orc.runtime.nodes.Node)
     */
    public Node compile(Node output) {
        Node node = right.compile(output);
        if (var != null && var.length() > 0)
            node = new Assign(var, node);
        if (publish)
            node = new Fork(node, output);
        return left.compile(node);
    }

    public String toString() {
        return "{" + left + "\n>" + (publish ? "!" : "") + var + "> " + righ
t + "}";
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.ast;

import java.util.ArrayList;
import java.util.List;

import orc.runtime.nodes.Node;
import orc.runtime.nodes.Store;
import orc.runtime.nodes.Where;
import orc.runtime.values.Tuple;

/**
 * @author wcook
 *
 * Abstract syntax for "where" expression
 */
public class AsymmetricParallelComposition extends OrcProcess {
    /**
     * The body in the form
     * <pre>
     *     body where bindings
     * </pre>
     */
    OrcProcess body;

    /**
     * The bindings in the form
     * <pre>
     *     body where bindings
     * </pre>
     */
    List<Binding> bindings = new ArrayList<Binding>();

    public AsymmetricParallelComposition(OrcProcess body) {
        this.body = body;
    }

    public void addBinding(String name, OrcProcess item) {
        bindings.add(new Binding(name, item));
    }

    /**
     * Compiles the bindings and the body.
     * Most of the work is done by the bindings.
     * @see orc.ast.OrcProcess#compile(orc.runtime.nodes.Node)
     */
    public Node compile(Node output) {
        Node result = body.compile(output);
        for (Binding b : bindings) {
            result = b.compile(result);
        }
        return result;
    }

    public String toString() {
        return "{" + body + "\nwhere" + Tuple.format(’ ’, bindings, ";\n  ",
’}’);
    }

    class Binding {
        public String name;
        public OrcProcess item;
```

```
        public Binding(String name, OrcProcess item) {
            this.name = name;
            this.item = item;
        }

        /**
         * The item is compiled to output its result to a store node.
         * A Where node is created to run the binding and the body in para
llel.
         * @param base  node of the left side of the where
         * @return      returns node for complete where expression
         */
        public Node compile(Node base) {
            return new Where(base, name, item.compile(new Store(name)));
        }

        public String toString() {
            return name + "=" + item;
        }
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.ast;

import java.util.List;

import orc.runtime.nodes.Node;
import orc.runtime.nodes.Return;
import orc.runtime.values.Tuple;

/**
 * Abstract syntax for defintions (which can be nested) with the form
 * <pre>
 *    def name(formals) = body
 *    rest
 * </pre>
 * Where rest is the program in which the name is bound
 * @author wcook
 */
public class Define extends OrcProcess {

    String name;
    List<String> formals;
    OrcProcess body;
    OrcProcess rest;

    public Define(String name, List<String> formals, OrcProcess body,
            OrcProcess rest) {
        this.name = name;
        this.formals = formals;
        this.body = body;
        this.rest = rest;
    }

    /**
     * Compiles the body with output to a return node.
     * Creates a define node (which will created the binding) and
     * then invoke the rest of the program.
     * @see orc.ast.OrcProcess#compile(orc.runtime.nodes.Node)
     */
    public Node compile(Node output) {
        Node bodyNode = body.compile(new Return());
        Node restNode = rest.compile(output);
        return new orc.runtime.nodes.Define(name, formals, bodyNode, restN
ode);
    }

    public String toString() {
        return "def " + name + Tuple.format('(', formals, ",", ')') + " =\n
"
                + body + "\n" + rest;
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.ast;

import java.util.ArrayList;
import java.util.List;

import orc.runtime.nodes.Node;
import orc.runtime.nodes.Param;
import orc.runtime.values.Tuple;


/**
 * Abstract syntax for calls. Includes both site calls and definition call
s
 * @author wcook
 */
public class Call extends OrcProcess {
    String name;
    List<OrcProcess> args;

    static int varNum;

    public Call(String name, List<OrcProcess> args) {
        this.name = name;
        if (args == null)
            args = new ArrayList<OrcProcess>();
        this.args = args;
    }

    /**
     *
     * @see orc.ast.OrcProcess#compile(orc.runtime.nodes.Node)
     */
    public Node compile(Node output) {
        for (OrcProcess p : args)
            if (p.asParam() == null)
                return translate().compile(output);

        List<Param> params = new ArrayList<Param>();
        for (OrcProcess p : args)
            params.add(p.asParam());

        return new orc.runtime.nodes.Call(name, params, output);
    }

    /**
     * Translates nested calls:
     * <pre>
     *    M(A(), B())
     * </pre>
     * is interpreted as
     * <pre>
     *    M(a, b) where a = A(); b = B()
     * </pre>
     * @return process to be executed
     */
    public OrcProcess translate() {
        List<OrcProcess> newArgs = new ArrayList<OrcProcess>();

        Call newCall = new Call(name, newArgs);
        AsymmetricParallelComposition where =
            new AsymmetricParallelComposition(newCall);
        for (OrcProcess p : args)
            if (p.asParam() != null)
                newArgs.add(p);
            else {
                String newVar = "V" + varNum++;
                newArgs.add(new Variable(newVar));
                where.addBinding(newVar, p);
            }

        return where;
    }

    public String toString() {
        return name + Tuple.format('(', args, ",", ')');
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.ast;

import orc.runtime.nodes.Node;
import orc.runtime.nodes.Param;

/**
 * Abstract syntax for literals
 * @author wcook
 */
public class Literal extends OrcProcess {
    public Object value;

    public Literal(Object value) {
        this.value = value;
    }

    public String toString() {
        if (value instanceof String)
            return "\"" + value + "\"";
        else
            return value.toString();
    }

    /**
     * Creates a literal node
     * @see orc.ast.OrcProcess#compile(orc.runtime.nodes.Node)
     */
    public Node compile(Node output) {
        return new orc.runtime.nodes.Literal(value, output);
    }

    /**
     * When used as a parameter, the literal just outputs its value.
     * @see orc.ast.OrcProcess#asParam()
     */
    public Param asParam() {
        return new orc.runtime.nodes.Literal(value, null);
    }
}
```

```
/*
 * Copyright 2005, The University of Texas at Austin. All rights reserved.
 */
package orc.ast;

import orc.runtime.nodes.Node;
import orc.runtime.nodes.Param;

/**
 * Abstrac syntax for variables
 * @author wcook
 */
public class Variable extends OrcProcess {
    String var;

    public Variable(String var) {
        this.var = var;
    }

    /**
     * When used as a parameter, creates a variable node to look up the va
lue.
     * @see orc.ast.OrcProcess#asParam()
     */
    public Param asParam() {
        return new orc.runtime.nodes.Variable(var);
    }

    /**
     * Cannot be used as a process. That is "x" alone is not a valid Orc p
rogram.1
     * @see orc.ast.OrcProcess#compile(orc.runtime.nodes.Node)
     */
    public Node compile(Node output) {
        throw new Error("Only used as a parameter");
    }

    public String toString() {
        return var;
    }
}
```