

### Introduction

Metropolis framework

- Methodology
- Meta model of computation
- Language: Meta model
  - function specification
  - design constraints
  - architecture specification
- Formal methods

1

CS269: HW/SW Engineering of Embedded Systems, Winter02

### Introduction

2

CS269: HW/SW Engineering of Embedded Systems, Winter02

### Design criteria

- Code reuse
  - Based on the Java parser of Ptolemy II (well, sort of...)
- Reduce effort to develop back-end modules
  - APIs to inspect the ASTs
  - A generic traversal mechanism: Visitor design pattern
  - An annotation mechanism: Properties
- Support incremental compilation (to be implemented)
  - Transparent from the point of view of the user
  - Reuse ASTs from previous compilations
  - Generate "skeletons" Meta-model files

3

CS269: HW/SW Engineering of Embedded Systems, Winter02

### The implementation

- Language: Java (1.2 or later)
- Development: Unix
- Sources (CVS): metro/src/
- Packages:
 

metropolis/metamodel	Main package of the compiler
metropolis/metamodel/frontend	Front-end of the compiler
metropolis/metamodel/nodetypes	Definition of the nodes of the AST
metropolis/metamodel/backends	Back-end modules of the compiler

4

CS269: HW/SW Engineering of Embedded Systems, Winter02

### Overall compilation flow

Front - end

Loads meta-model files  
Checks syntax and semantics  
Builds ASTs + semantic information

Elaboration

Get the static structure of the network: # of nodes and their connections  
Needed by backends that need static information about the network

Back - end

Traverse and analyze the ASTs  
Emits an output (languages, formal methods)  
Several back-ends, user should select the back-end to be used

5

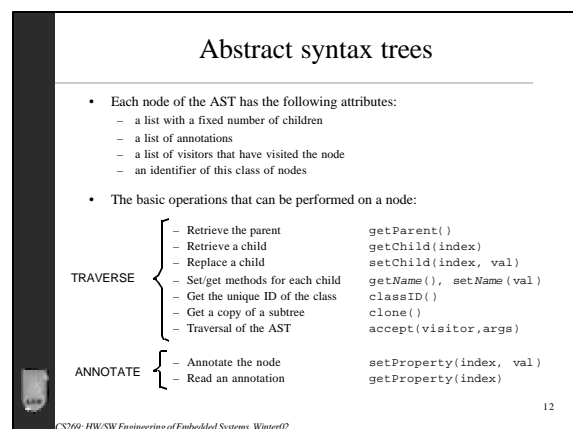
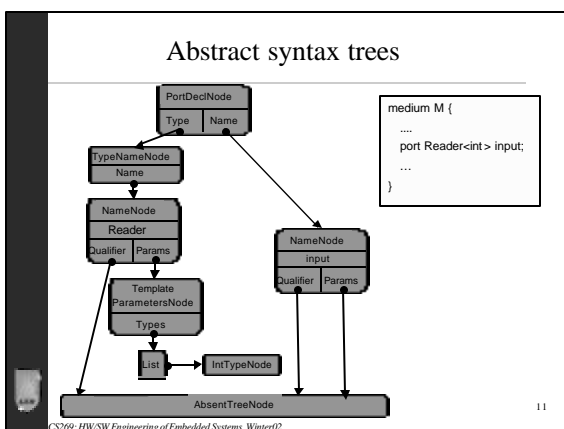
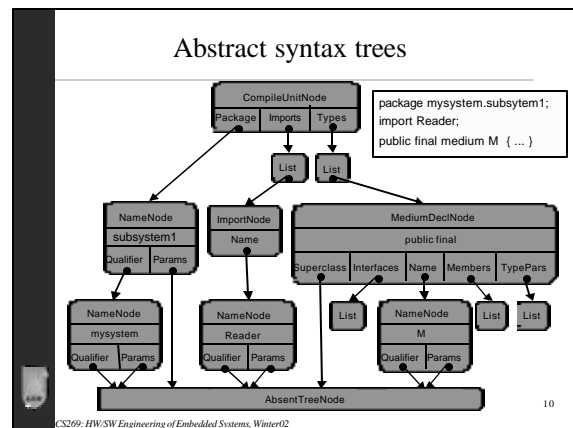
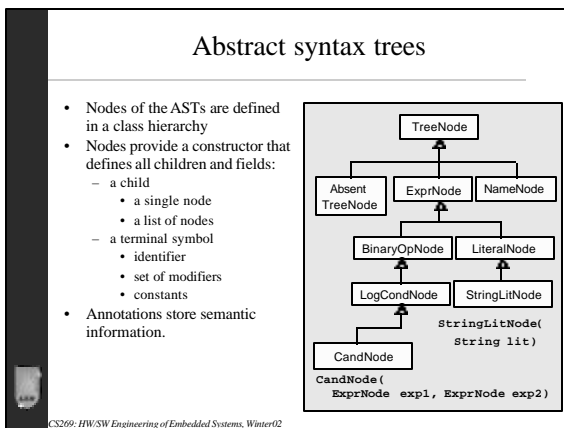
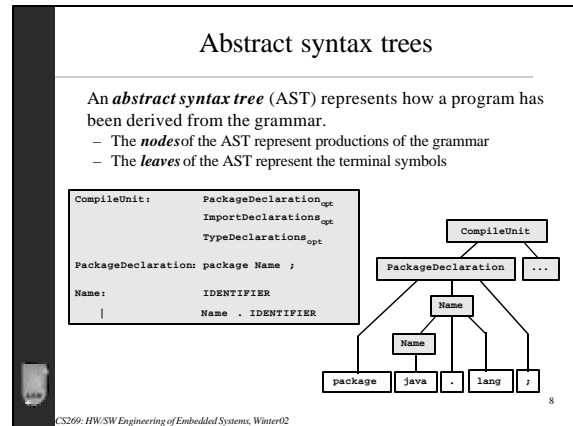
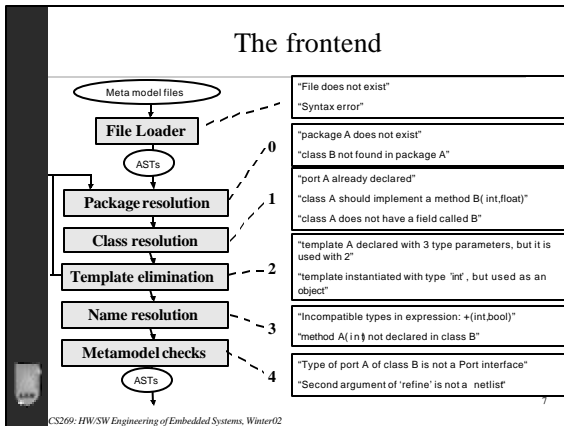
CS269: HW/SW Engineering of Embedded Systems, Winter02

### The frontend

- All file accesses and incremental compilation are encapsulated in FileLoader:
  - loads files from disk
  - check syntax of the program
  - builds the ASTs
- loadCompileUnit(filename, pass)
- loadCompileUnit(pkg, class, pass)
- Tasks of semantic passes
  - check semantics of the program
  - add semantic annotations
  - refine the abstract syntax trees, e.g. Name → FieldAccess

6

CS269: HW/SW Engineering of Embedded Systems, Winter02



### Abstract syntax trees

Example: annotate the AST of a file with the number of classes declared in that file.

```

// Identifier of our property
public static final int NUMDECL_KEY = RESERVED_PROPERTIES + 1;

// Method to perform the annotation
void annotateNumDecls (CompileUnitNode ast {
    int numDecls = ast.getDefTypes().size();
    ast.setProperty (NUMDECL_KEY, new Integer (numDecls));
}

// Method to read the annotation
int getNumDecls (CompileUnitNode ast) {
    Integer num = (Integer) getProperty (NUMDECL_KEY, val);
    return num.intValue();
}
    
```

CS269: HW/SW Engineering of Embedded Systems, Winter02 13

### Annotations

The front-end provides several annotations with semantic information

- On the `CompileUnitNode`: information about the file
 

IDENT_KEY	File name of the compile unit
PACKAGE_KEY	Enclosing package
IMPORTED_PACKAGES_KEY	List of imported packages
- On other nodes: specific informations
 

DECL_KEY	Name Node	Declaration of the name
THIS_CLASS_KEY	ThisNode	TypeNode of the class
SUPERCLASS_KEY	ClassDeclNode	Superclass of this type
JUMP_DESTINATION_KEY	Jump nodes	Destination of a jump
TYPE_KEY	ExprNode	Type of the expression

CS269: HW/SW Engineering of Embedded Systems, Winter02 14

### Declarations

```

public update void write(int val) {
    buffer[current++] = val;
}
    
```

- Each node that declares a new name is annotated with a unique object, the **declaration**.
- Each reference to a name is annotated with the unique declaration of that name.

CS269: HW/SW Engineering of Embedded Systems, Winter02 15

### Declarations: APIs

- Basic information:
  - name: Identifier of this declaration
  - category: Kind of declaration (e.g. CG\_PROCESS)
- Optional information:
  - modifiers: Set of modifiers (if any)
  - container: Enclosing declaration (if any)
  - source: Node of the AST where the declaration is performed (if any)
  - scope: Symbol table with names declared inside this declaration (if any)
- Each information defines methods `hasInfo()`, `getInfo()` and `setInfo()` e.g.: `setModifiers()`, `getName()`

CS269: HW/SW Engineering of Embedded Systems, Winter02 16

### Declarations: APIs

Additional methods allow us to get specific information from the declarations, without traversing the AST.

- In `PackageDecl`:
  - Get all subpackages of this package: `getSubPackages()`
  - Get subpackage with a given name: `getSubPackage (name)`
  - Get all types defined in this package: `getUserTypes ()`
  - Get an type with a given name: `getUserType (name)`
- In `ObjectDecl`
  - Get the superclass: `getSuperClass ()`
  - Get the interfaces implemented: `getInterfaces ()`
  - Get all fields/one field: `getFields()/getField (name)`
  - Get all ports/one port: `getPorts ()/getPort (name)`
  - Get all parameters/ only one: `getParameters ()/getParameter (name)`
  - Get all labels/ only one: `getLabels ()/ getLabel (name)`
  - Get all methods/all with a name: `getMethods ()/getMethods (name)`
  - Get all constructors: `getConstructors ()`

CS269: HW/SW Engineering of Embedded Systems, Winter02 17

### Visitors

- Traversals of the AST use the Visitor design pattern:
  - "A solution to the problem of adding operations to the elements of an object structure without changing the classes on which it operates"
  - 1 traversal = 1 Visitor class
  - AST classes not modified when adding new Visitors
- The code of the traversal is divided in `visit ()` methods:
 

```

Object visitNameNode (NameNode node,
                    LinkedList args)
            
```
- There is one `_defaultVisit ()` method used if `visit ()` method is not defined for a class of nodes
- Back-end tool may need several traversals of the AST
  - Back-end tool = set of visitors + additional classes

CS269: HW/SW Engineering of Embedded Systems, Winter02 18

### Visitors

```

public class MetaModelVisitor {
    public MetaModelVisitor(int traversalMethod) { ... }
    public Object visitCompileUnitNode(CompileUnitNode node, LinkedList args) {
        return _defaultVisit(node, args);
    }
    public Object visitProcessDeclNode(ProcessDeclNode node, LinkedList args) {
        return _defaultVisit(node, args);
    }
    ...
    protected Object _defaultVisit( TreeNode node, LinkedList args) {
        return null;
    }
}

public class MyVisitor extends MetaModelVisitor {
    // Implement visit methods and choose traversal method
}
    
```

How to traverse the tree  
e.g. preorder

What to do on each node  
e.g. emit code

19

CS269: HW/SW Engineering of Embedded Systems, Winter02

### How to write a back-end?

```

public interface Backend {
    void invoke(Listargs, List sources);
}
    
```

STEPS

1. Write a class that implements Backend
2. Write the method invoke() in that class
  - define the arguments
  - design necessary visitors
3. Write visitors to traverse the ASTs
4. Add flags to the compiler command-line

20

CS269: HW/SW Engineering of Embedded Systems, Winter02

### How to write a back-end?

```

public interface Backend {
    void invoke(Listargs, List sources);
}

public class SimulatorBackend implements Backend {
    public abstract void invoke(Listargs, List sources) {
    }
}
    
```

STEPS

1. Write a class implements Backend
2. Write method invoke() in that class
  - define the arguments
  - design necessary visitors
3. Write visitors to traverse the ASTs
4. Add flags to the compiler command-line

21

CS269: HW/SW Engineering of Embedded Systems, Winter02

### How to write a back-end?

```

public interface Backend {
    void invoke(Listargs, List sources);
}

public class SimulatorBackend implements Backend {
    public abstract void invoke(Listargs, List sources) {
        List flags = inspectArguments (args);
        Iterator iter = sources.iterator();
        while (iter.hasNext()) {
            CompileUnitNode ast = iter.next();
            String code = ast.accept(new SimulatorVisitor(), flags);
            printSimulationCode(code);
        }
    }
}
    
```

STEPS

1. Write a class implements Backend
2. Write method invoke() in that class
  - define the arguments
  - design necessary visitors
3. Write visitors to traverse the ASTs
4. Add flags to the compiler command-line

22

CS269: HW/SW Engineering of Embedded Systems, Winter02

### How to write a back-end?

```

public interface Backend {
    void invoke(Listargs, List sources);
}

public class SimulatorBackend implements Backend {
    public abstract void invoke(Listargs, List sources) {
        List flags = inspectArguments (args);
        Iterator iter = sources.iterator();
        while (iter.hasNext()) {
            CompileUnitNode ast = iter.next();
            String code = ast.accept(new SimulatorVisitor(), flags);
            printSimulationCode(code);
        }
    }
}

public class SimulatorVisitor extends MetaModelVisitor {
    public Object visitProcessDeclNode(
        ProcessDeclNode node, Listargs) { ... }
    public Object visitAwaitStatementNode(
        AwaitStatementNode node, List args) { ... }
    ...
}
    
```

STEPS

1. Write a class implements Backend
2. Write method invoke() in that class
  - define the arguments
  - design necessary visitors
3. Write visitors to traverse ASTs
4. Add flags to compiler command-line

23

CS269: HW/SW Engineering of Embedded Systems, Winter02

### How to write a back-end?

```

protected static Backend backend = null;
protected static ListbackendArgs = new LinkedList();
protected static String[] helpMessage = {" ", ""};
protected static String[] usageMessage = {" ", ""};

public class Compiler {
    public static void main (String[] args) {
        inspectArgs(args);
        initCompiler();
        asts = compileSources();
        backend.invoke( backendArgs, asts );
    }
    public static void inspectargs(String[] args) {
    }
}
    
```

STEPS

1. Write a class implements Backend
2. Write method invoke() in that class
  - define the arguments
  - design necessary visitors
3. Write visitors to traverse ASTs
4. Add flags to compiler command-line

24

CS269: HW/SW Engineering of Embedded Systems, Winter02

### How to write a back-end?

```
protected static Backend backend = null;
protected static ListBackendArgs = new LinkedList();
protected static String[] helpMessage = {"", ""};
protected static String[] usageMessage = {"", ""};
```

```
public class Compiler {
    public static void main (String[] args){
        inspectArgs(args);
        initCompiler();
        asts = compileSources();
        backend.invoke(backendArgs, asts);
    }
    public static void inspectArgs(String[] args){
        for (int i = 0; i < args.length; i++) {
            String current = args[i];
            ....
            if (current.equals("-simulator")) {
                backend = new SimulatorBackend();
                backendArgs = ....;
            }
        }
    }
}
```

**STEPS**

1. Write a class implements Backend
2. Write method invoke() in that class
  - define the arguments
  - design necessary visitors
3. Write visitors to traverse ASTs
4. Add flags to compiler command-line

CS269: HW/SW Engineering of Embedded Systems, Winter02

### Documentation of the code



CS269: HW/SW Engineering of Embedded Systems, Winter02

## Writing a Visitor

27

CS269: HW/SW Engineering of Embedded Systems, Winter02

### Steps to Write a Visitor (1)

- What the return value is?
  - If new information only needs to be stored in the nodes of the AST, return null in this case.
  - Usually, the return value for all the visit methods are consistent.
- What nodes in the AST to be visited?
  - only a few nodes
  - all nodes

These will determine which traversal method to use.

28

CS269: HW/SW Engineering of Embedded Systems, Winter02

### Steps to Write a Visitor (2)

- Which traversal method to use?
 

There are 3 types of traversal:

  - TM\_CUSTOM
    - explicit calls need to be made to visit a particular node
    - Useful when only a few nodes in the AST need to be visited.

29

CS269: HW/SW Engineering of Embedded Systems, Winter02

### Steps to Write a Visitor (3)

- TM\_CHILDREN\_FIRST
  - visit children, then visit myself. Return value can be stored in the parent node.
  - if all nodes need to be visited and the return value of the children will be needed by the parent.
- TM\_SELF\_FIRST
  - visit myself, then visit children
  - Used if all nodes need to be visited and the return value of the children is not required.

30

CS269: HW/SW Engineering of Embedded Systems, Winter02

### Structure of a Visitor (1)

File: MetaModelBackend.java

```

package metropolis.metamodel.backends;

import metropolis.*;
import metropolis.metamodel.*;
import metropolis.metamodel.nodetypes.*;

public class MetaModelBackend extends MetaModelIVisitor {
    public MetaModelBackend() {
        super(TM_CUSTOM); // traversal method
    }
}
    
```

31

CS269: HW/SW Engineering of Embedded Systems, Winter02

### Structure of a Visitor (2)

```

public Object visitCompileUnitNode
    (ConstraintBlockNode node,
     LinkedList args) {
    return null;
}

public Object visitLabeledBlockNode (LabeledBlockNode node,
    LinkedList args) {
    return null;
}

protected Object _defaultVisit(TreeNode node, LinkedList args) {
    return null;
}
    
```

32

CS269: HW/SW Engineering of Embedded Systems, Winter02

### What to do with a node? (1)

```

block (labelA) {
    stmt1;
    stmt2;
}
    
```

33

CS269: HW/SW Engineering of Embedded Systems, Winter02

### What to do with a node? (2)

Look at LabeledBlockNode.java under:  
metropolis/metamodel/nodetypes:

```

public static final int CHILD_INDEX_STMTS = 0;
public static final int CHILD_INDEX_LABEL = 1;

public final List getStmts() { ... }
public final LocalLabelNode getLabel() { ... }
    
```

Children of node can be retrieved by:

```

node.getStmts();
node.getLabel();
    
```

34

CS269: HW/SW Engineering of Embedded Systems, Winter02

### Using TM\_CUSTOM (1)

- To traverse a list of nodes, use this static method:  

```

ArrayList TNLManip.traverseList
    (IVisitor visitor, LinkedList args, List list);
        
```
- Example:

```

public Object visitBlockNode(BlockNode node, LinkedList args)
{
    TNLManip.traverseList(this, args, node.getStmts()); //or
    List retValue = TNLManip.traverseList(this, args, node.getStmts());
}
        
```

35

CS269: HW/SW Engineering of Embedded Systems, Winter02

### Using TM\_CUSTOM (2)

- To traverse only a node at one time, invoke this method on at TreeNode:

```

Object accept(IVisitor visitor, LinkedList args);
        
```
- Example:

```

public Object visitLabeledBlockNode(BlockNode node, LinkedList args)
{
    node.getLabel().accept(this, args);
}
        
```

36

CS269: HW/SW Engineering of Embedded Systems, Winter02

### Using TM\_CHILDREN\_FIRST

- One useful method to retrieve the return values of the children nodes:
  - Object childReturnValueAt(int index);

```

    graph TD
      BlockNode[BlockNode] --> StatementNode1[StatementNode]
      BlockNode --> StatementNode2[StatementNode]
    
```

- Example:
 

```

                public Object visitBlockNode(BlockNode node, LinkedList args){
                return node.childReturnValueAt(node.CHILD_INDEX_STMTS)
                }
            
```

CS269: HW/SW Engineering of Embedded Systems, Winter02 37

### MetaModelCodeGenVisitor.java

(EXAMPLE)

```

    package metropolis.metamodel.backends.metamodel;

    import metropolis.metamodel.*;
    import metropolis.metamodel.nodetypes.*;

    public class MetaModelCodeGenVisitor extends MetaModelVisitor
    implements MetaModelStaticSemanticConstants{

    public MetaModelCodeGenVisitor() {
        super(TM_CUSTOM);
    }
    
```

CS269: HW/SW Engineering of Embedded Systems, Winter02 38

### MetaModelCodeGenVisitor.java

```

    public Object visitCompileUnitNode (CompileUnitNode node,
    LinkedList args) {
    LinkedList retList = new LinkedList();

    if (node.getPkg() != AbsentTreeNode.instance) {
        retList.addLast ("package ");
        retList.addLast (node.getPkg().accept(this, args));
        retList.addLast (";\n");
    }
    retList.addLast (TNLManip.traverseList (this, args, node.getImports()));
    retList.addLast ("\n");
    Iterator typeIter = TNLManip.traverseList
    (this, args, node.getDefTypes()).iterator();
    ...
    return _stringListToString(retList);
    }
    
```

CS269: HW/SW Engineering of Embedded Systems, Winter02 39

### Other Useful Methods

- StringManip.java
- TNLManip.java

CS269: HW/SW Engineering of Embedded Systems, Winter02 40

### Other Useful Methods (1)

- To create a list containing one object:
 

```
LinkedList TNLManip.addFirst(Object obj)
```
- Examples:
  - Preparing a list of one string as an argument:
 

```
node.getName ().accept(this, TNLManip.addFirst("an object"));
```
  - Preparing a list with one string as return value:
 

```
return TNLManip.addFirst("return object");
```

CS269: HW/SW Engineering of Embedded Systems, Winter02 41