An Lightweight Infrastructure to Support Experimenting with Heterogeneous Transformations

An Application of .NET

Wolfgang Lohmann, Günter Riedewald, Thomas Zühlke

University of Rostock, Germany

Outline

- Motivation
- Example
- Trane
- OOTM
- Concluding remarks

Motivation

- Transformation of software (T:I*→ O+)
  - Examples: Compiler, XSLT-scripts, data format converters, source-to-source transformers, Unix tools like awk and sed
  - Applications in S2S: e.g. refactoring, maintenance, translation, software evolution through transformation
- Complex transformations
  - often developed in explorative way
  - different combinations and sequences are compared
  - intermediate results are queried

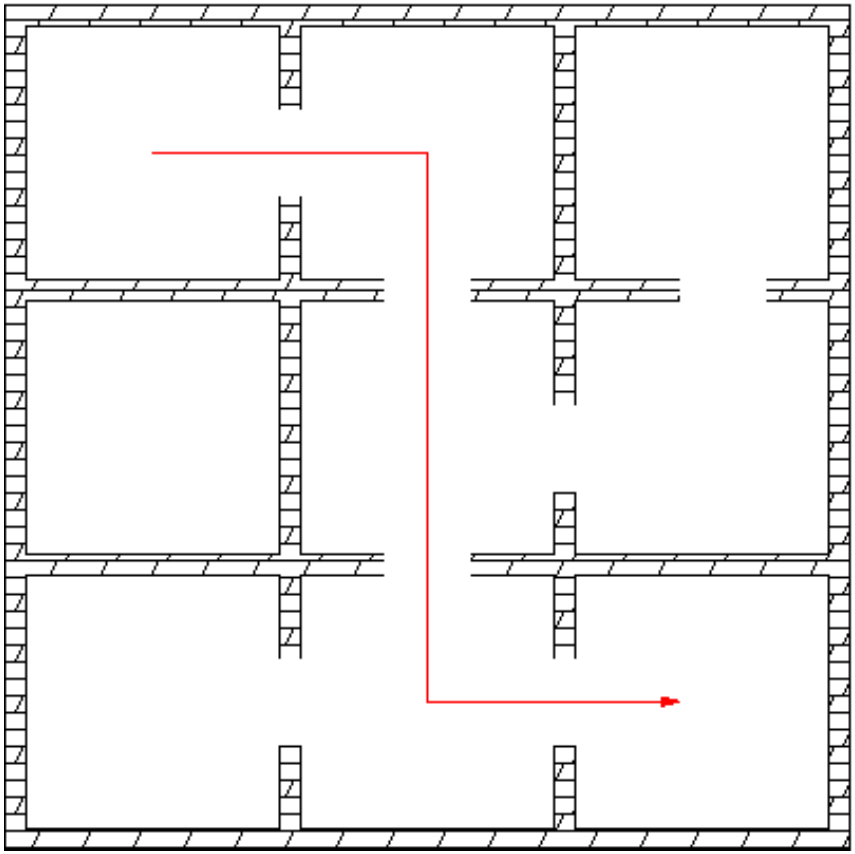
→ Support experiments

Motivation (2)

- Transformations of Language Artefacts
  - Grammars, semantic descriptions, components of language processors
  - Tools exist in and use different formats, e.g.
    - GDK, MetaEnvironment, Stratego, TXL
    - Different grammar formalisms (ATerms, EBNF, XML ..)
    - Command line tools,
    - Web services
    - Algorithms in C, Prolog
    - DLLs, Plugins for Eclipse

→ Support combination of heterogeneous transformations

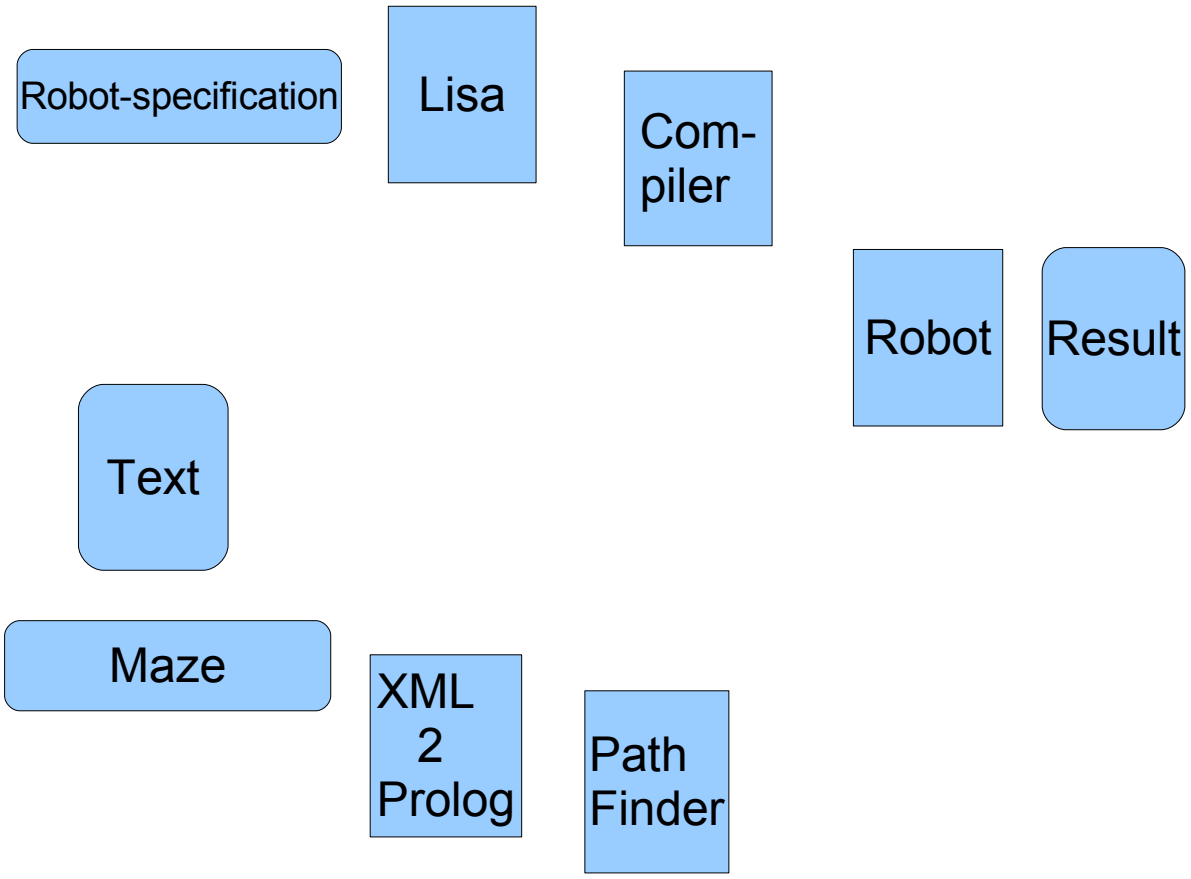
Imagine...



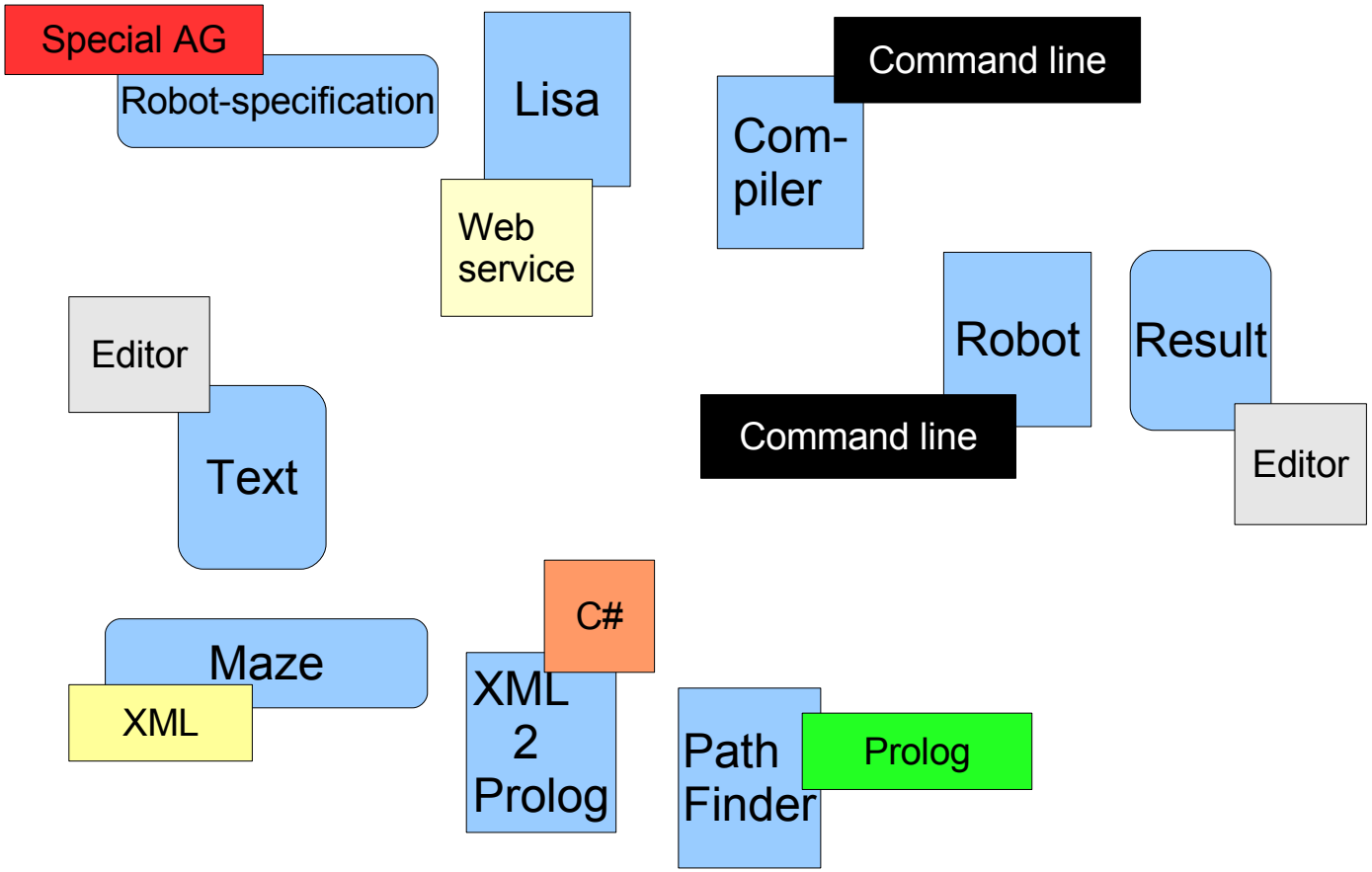
- Model of a company‘s building or a maze in a virtual world
- Robot to be controlled along a path
- Simple language to program robots

T : Problem → Program

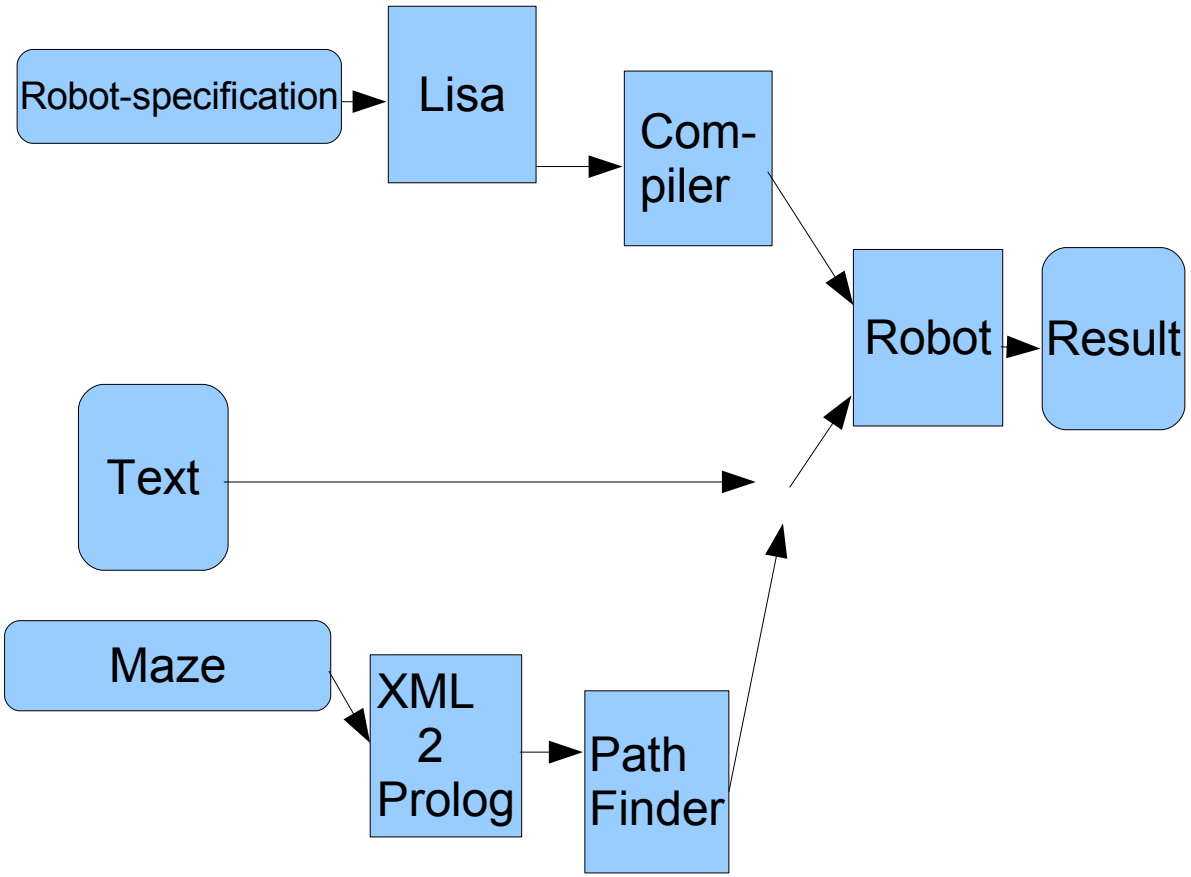
What we might have



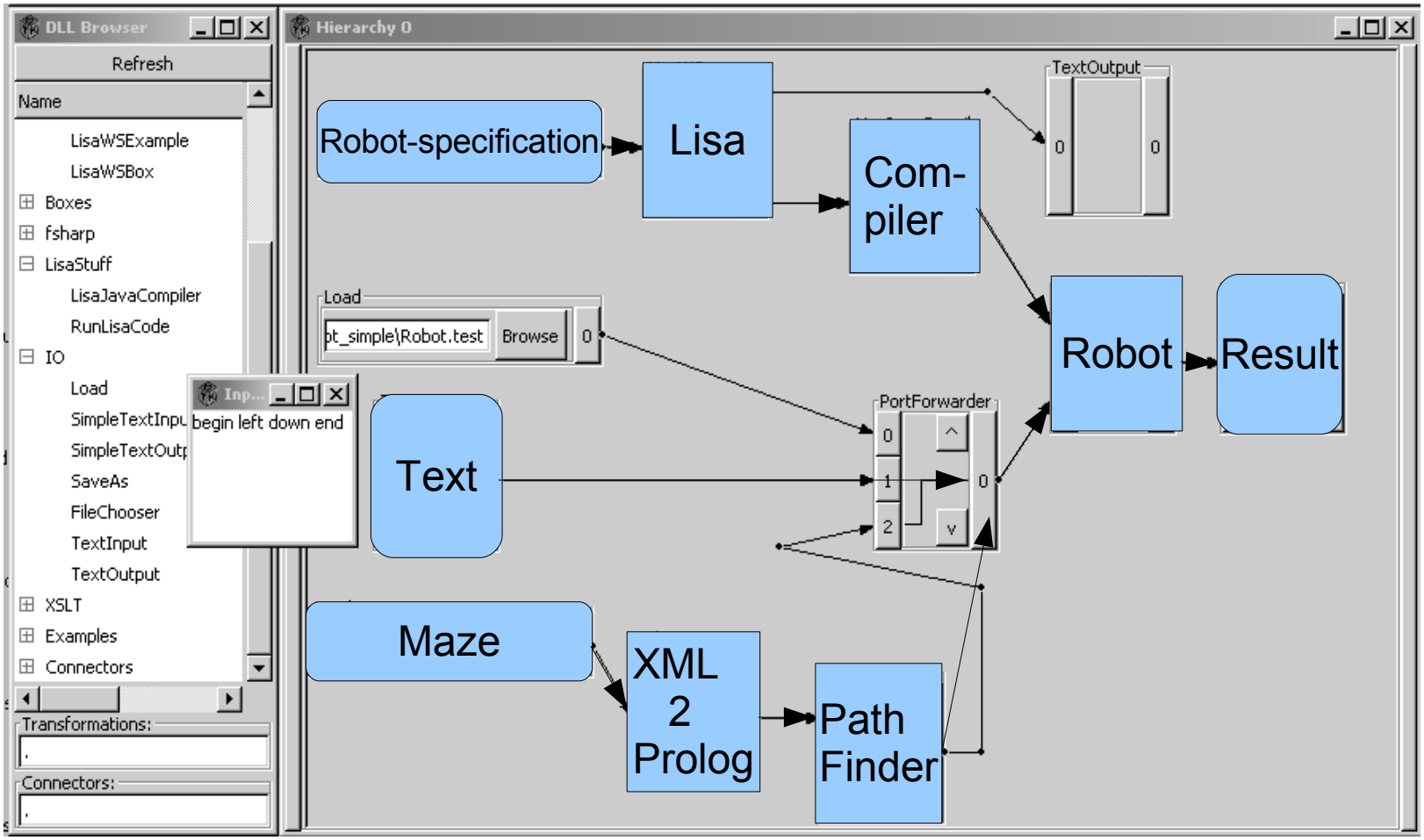
However...


Special AG
Robot-specification
Lisa
Web service
Command line
Com-piler
Robot
Command line
Result
Editor
Editor
Text
C#
Maze
XML
XML 2 Prolog
Path Finder
Prolog

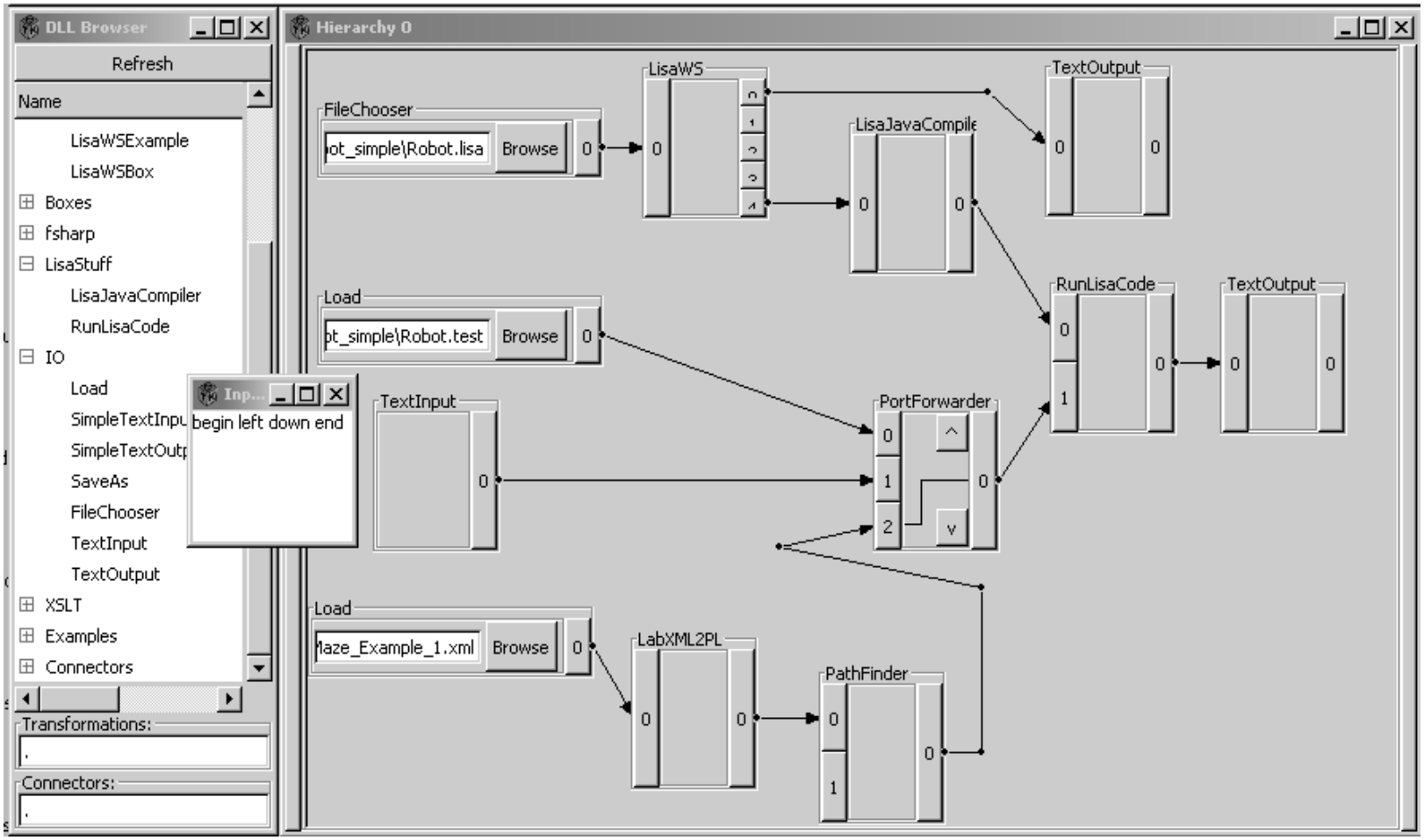
What we want to do



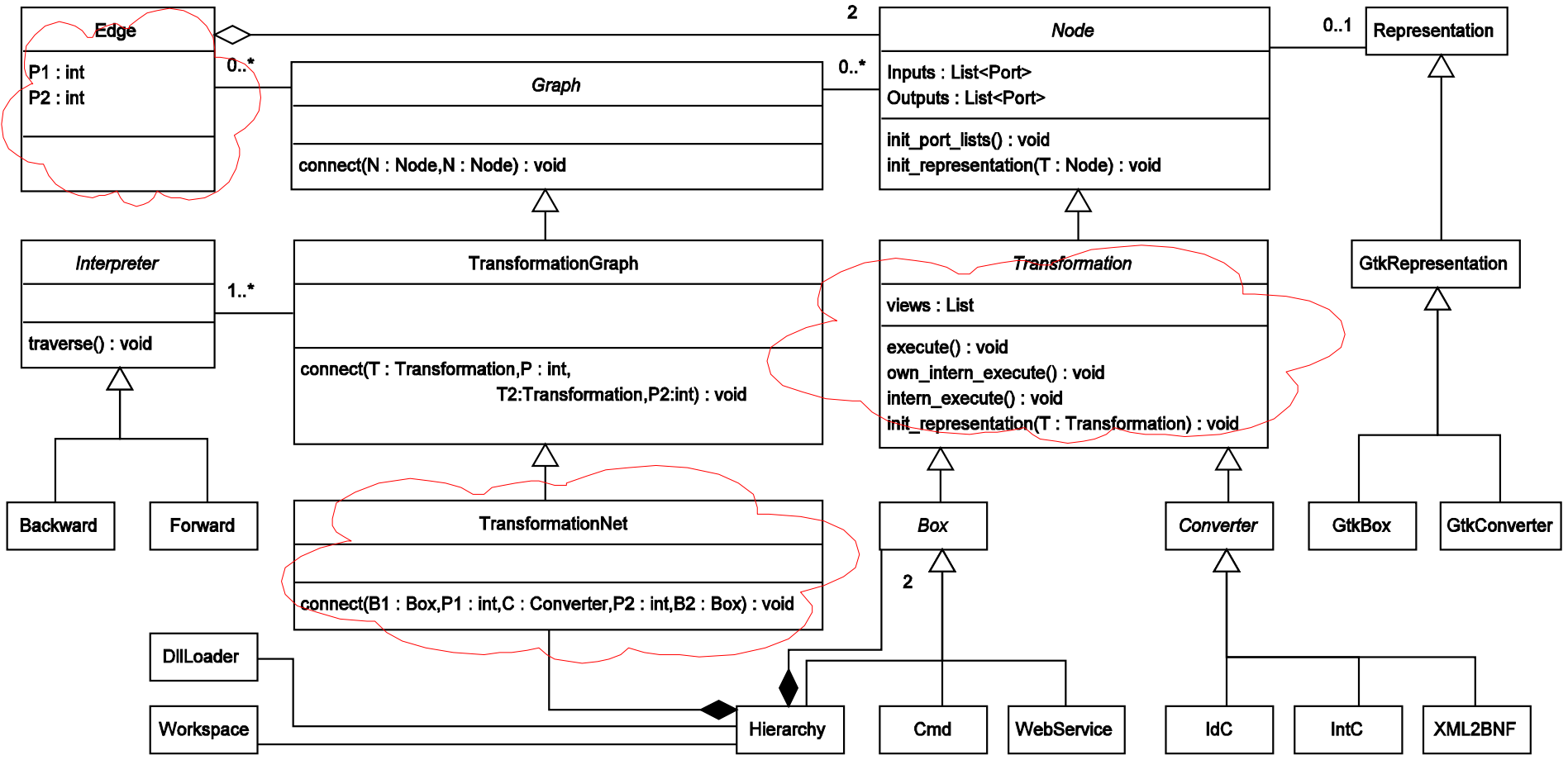
Transformation nets



Transformation nets



O-O Transformation Model



Different Kinds of Transformation, thanks to .NET

- Plain boxes : a simple C# class, which maps inputs to outputs with a function
- Web services : support for web service
- Command line tool wrappers : System.Diagnostics.Process
- Foreign Language Interface : DllImport(name)
- XSLT : XML-support
- F# transformations : cross language inheritance
- Hierarchy-boxes: Overriding of properties

Concluding remarks

.NET, and everything is ok?

- Plattform independent due to .NET?
  - Windows: MS .NET SDK, VS.NET, Gtk#, Mono,
    Linux: Mono and Gtk#
  - This application has a second level of P-Dependency:
    - the wrapped transformation has to be available
    - it has to be available on the same way
  - DllImport(„plwin.dll“) vs. „libpl.so“
- Cross-Language inheritance?
  - Core with C#
  - J# on Linux? F# access to properties of lists ?
  - Renaming scheme might lead to problems, e.g. Eiffel:
    AaBb becomes aa_bb

Concluding remarks

Summary

- OOTM with facilities to combine transformations and experiment, query intermediate results
- Wrapping of different 5 kinds of transformations behind a unique representation, others ?

The art is the wrapping, and this is easy due to .NET

Future Work

- Type system
  - What are types?
  - Explicite/ implicite casting
- Classification of boxes
- Usability
- Integration of Stratego, and many more boxes

Thank you!

Example: IntID-Box

```
public class IntID : Box
{
  public IntID()          {          }

  public override void init_port_lists()
  {
      Name = "IntID";
      Inputs.Add(new Port("int"));
      Inputs[0].data = new ValueData(0,"int");
      Outputs.Add(new Port("int"));
      Outputs[0].data = new ValueData(null,"int");
  }

  public override void execute()
  {
      Console.WriteLine("IntID-Box: start computation");
      Outputs[0].Data = Inputs[0].Data.copy();
      Console.WriteLine("IntID-Box: Done");
  }

  public override void initialise_representation()
  {
      this.Representation = Gtk_Box_Representation(this);
  }
}
```

- To create a box inherit from Box (at least from Transformation
- Configure name, input and and output ports
- Define transformation behaviour by overriding *execute*
- Define a representation (here not necessary)

XSLT and a derived box

```
public override void init_port_lists()
{
     Inputs.Add(new Port("string"));
     Inputs[0].data = new ValueData("", "string");
    Outputs.Add(new Port("string"));
    Outputs[0].data = new ValueData(null, "string");
}

public virtual void apply_xslt_to_input()
{
    String xml_input = (String)((Inputs[0].data.copy()).value);
    StringReader xml_input_reader = new StringReader(xml_input);

    XPathDocument xpath_document =
                new XPathDocument(xml_input_reader);
    XslCompiledTransform transformation =
                new XslCompiledTransform();
    StringReader xsl_script_reader = new StringReader(XsltScript());
    XmlTextReader xsl_script =
                new XmlTextReader(xsl_script_reader);
    transformation.Load(xsl_script);
    StringWriter xml_output_writer = new StringWriter();
    XPathNavigator document_navigator =
                xpath_document.CreateNavigator();
    transformation.Transform
                (document_navigator, null, xml_output_writer);
    Outputs[0].Data.Value = xml_output_writer.ToString();
}

public override void execute()
{
     apply_xslt_to_input();
}
```

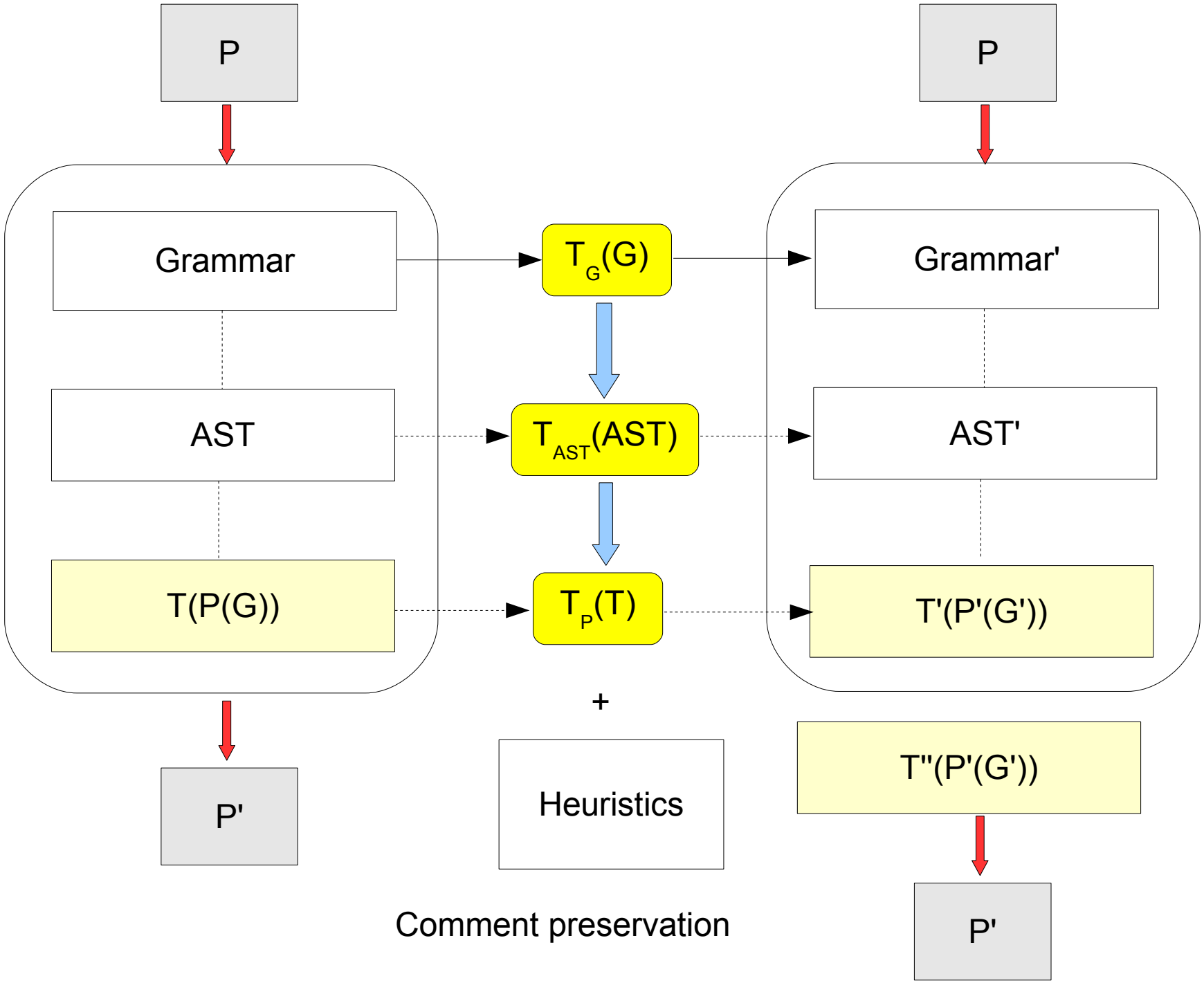
```
public class LabXML2PL : Apply_xslt_to_xml
{
   protected override String XsltScript() {
     String x =
@"<?xml version=""1.0""?>
<xsl:stylesheet version=""1.0""  xmlns:xsl=""http://www.w3.org/199
<xsl:output indent=""yes"" method=""text"" version=""4.0"" media-
<xsl:template match=""lab"">
  <xsl:apply-templates select=""room"" mode=""way-list""/>
</xsl:template>
<xsl:template match=""room"" mode= ""way-list"">
  <xsl:apply-templates select=""exit"">
  </xsl:apply-templates>
</xsl:template>
<xsl:template match=""exit"">
  way(<xsl:value-of select=""parent::node()/child::name""/>,
     <xsl:value-of select=""child::destination""/>,
     <xsl:value-of select=""child::direction""/>).
</xsl:template>
  </xsl:stylesheet>
";
      Console.WriteLine("XSLT Script: {0}", x);
      return x;
    }

public override void init_port_lists()
{
       base.init_port_lists();
       Name = "LabXML2PL";
}
```

```
using SbsSw.swi_pl_cs;
....
   public PathFinder()
   {
      String[] param = {@"H:\\....\\bin\\Debug\\Application.exe"};
      PlEngine e = new PlEngine(1, param);
      Path_to_Knowledge_Base = write_to_tmp_file(this.trafo());
   }

   public override void execute()
   {
      string Path_to_Application;
      string Path_to_Generated_Lab =
                     write_to_tmp_file((string)Inputs[0].Data.Value);
      string Path_to_Target       = (string)Inputs[1].Data.Value;
      String[] param = {@"H:\\..\\bin\\Debug\\Application.exe"  };
      PlTermv argument_vector =
                     new PlTermv(new PlTerm(Path_to_Knowledge_Base));
      PlQuery mq = new PlQuery("consult", argument_vector);
      bool success = mq.next_solution();
      mq.free();
      PlTermv argument_vector_2 =
                     new PlTermv(new PlTerm(Path_to_Generated_Lab));
      PlQuery consult_2 = new PlQuery("consult", argument_vector_2);
      success = consult_2.next_solution();
      consult_2.free();
      PlTermv argument_vector_3 = new PlTermv(2);
      argument_vector_3[0] = new PlTerm(Path_to_Target);
      PlQuery trafo = new PlQuery("main", argument_vector_3);
      success = trafo.next_solution();
      Console.WriteLine(argument_vector_3[1].ToString());
      trafo.free();
      Outputs[0].Data.Value = argument_vector_3[1].ToString();
      //"begin left down down right left left up end";
   }
```

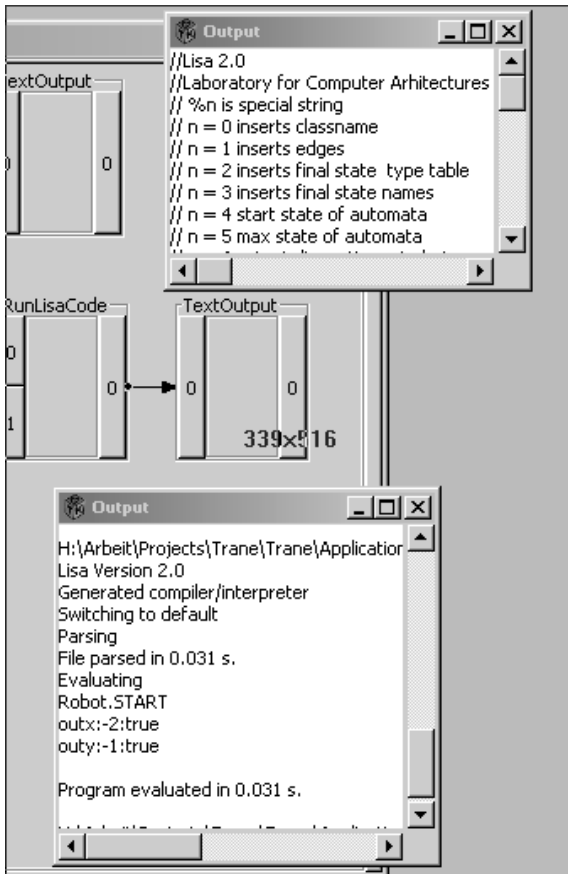
Prolog

- Use Swi_pl_cs.dll
- Inherit from box
- Start Prolog Engine
- Get inputs
- Depending on the types, construct Prolog calls

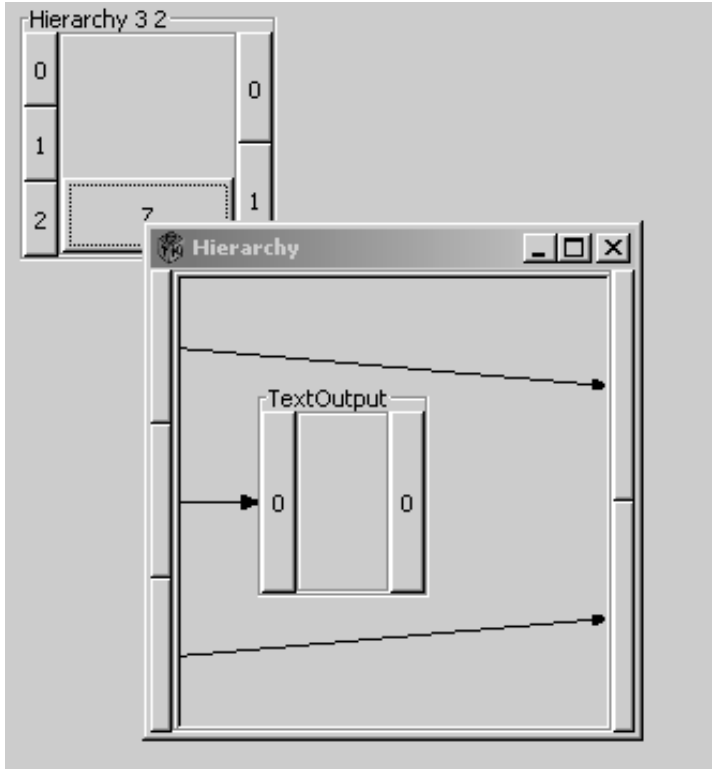
```
//Path_to_Knowledge_Base =   System.IO.Path.GetTempFileName();
            //Console.WriteLine(Path_to_Knowledge_Base);
            //System.IO.FileStream fs = new System.IO.FileStream(Path_to
            //System.IO.StreamWriter sw = new System.IO.StreamWriter(fs
            //sw.Write (this.trafo());
            //sw.Close();

 // instead of input, the filename:
         //string Path_to_Generated_Lab = (string)Inputs[0].Data.Value;
```


P
Grammar
AST
T(P(G))
P'
$T_G(G)$
$T_{AST}(AST)$
$T_P(T)$
+
Heuristics
P
Grammar'
AST'
T'(P'(G'))
T''(P'(G'))
P'
Comment preservation

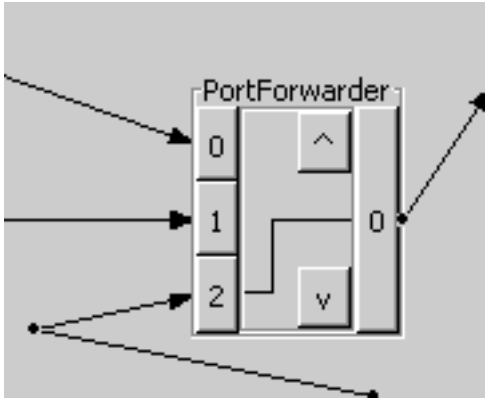
Text box


Output
//Lisa 2.0
//Laboratory for Computer Arhitectures
// %n is special string
// n = 0 inserts classname
// n = 1 inserts edges
// n = 2 inserts final state type table
// n = 3 inserts final state names
// n = 4 start state of automata
// n = 5 max state of automata
extOutput
0
RunLisaCode
0
1
0
TextOutput
0
0
339x516
Output
H:\Arbeit\Projects\Trane\Trane\Applicatior
Lisa Version 2.0
Generated compiler/interpreter
Switching to default
Parsing
File parsed in 0.031 s.
Evaluating
Robot.START
outx:-2:true
outy:-1:true
Program evaluated in 0.031 s.

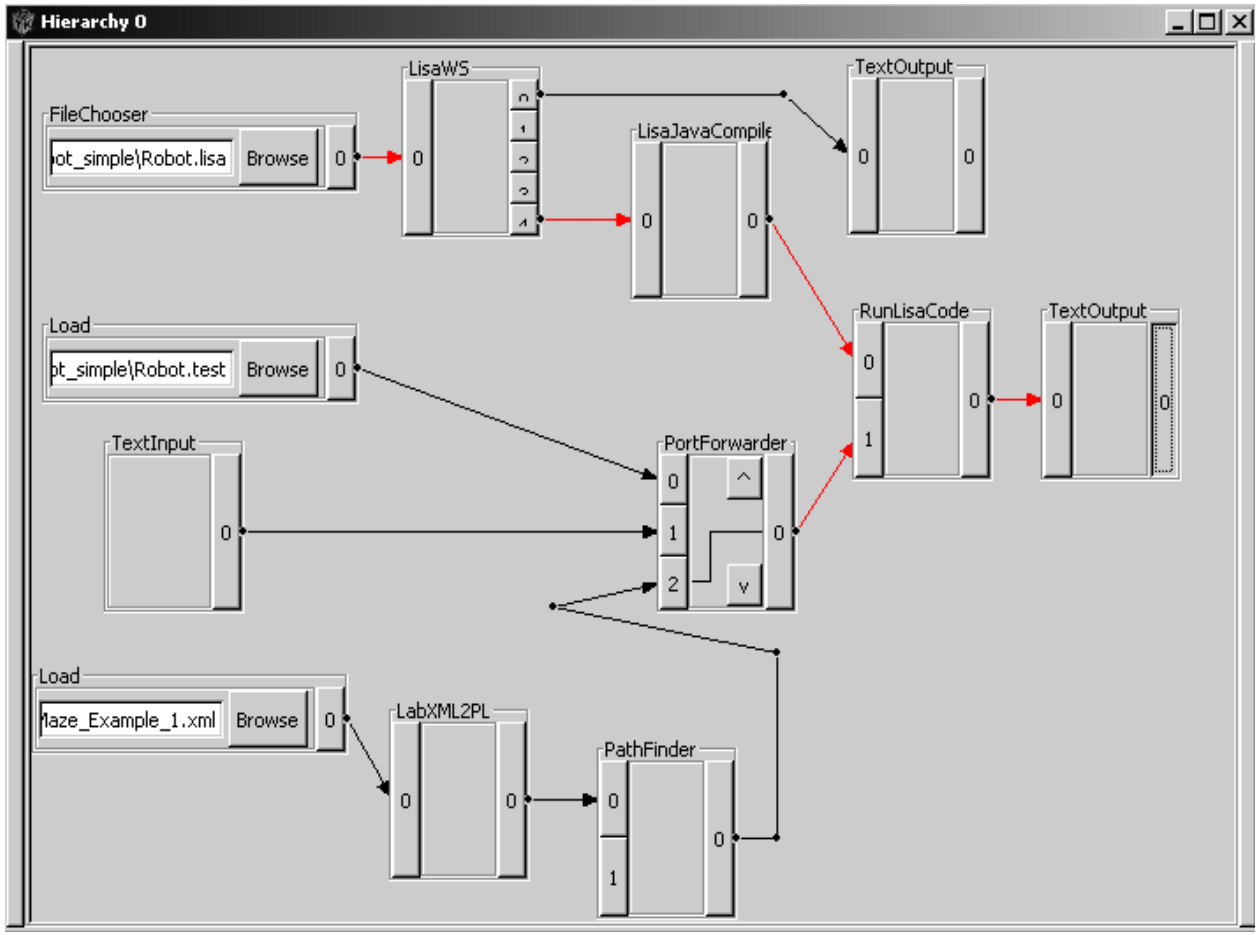
3:2 Hierarchy Box



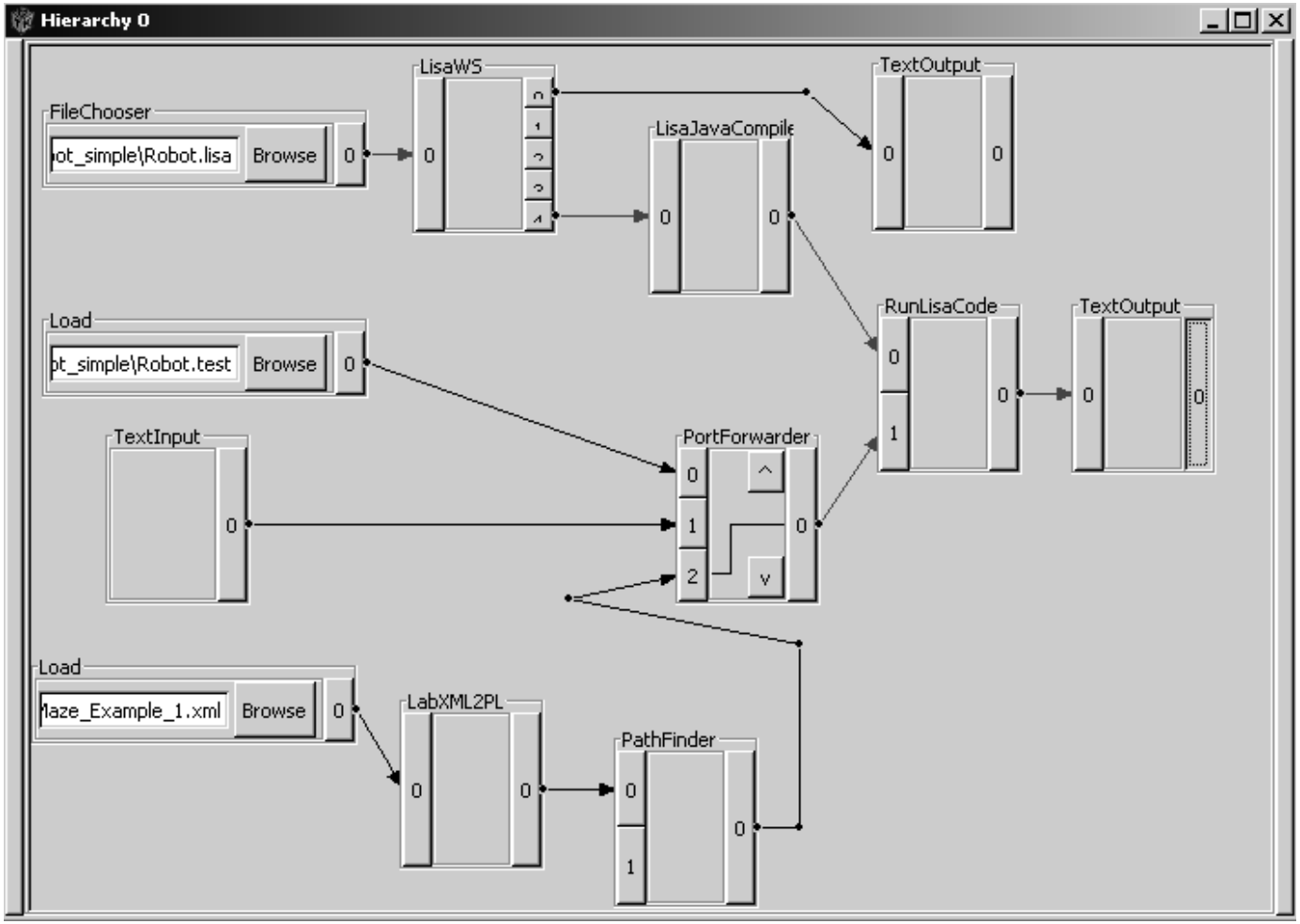
- Inherit from box
- Add an Id Box for both, Input and Output ports
- Override Properties to access to ports (refer to Id Boxes)
- Add representation with a net drawing area and generate lists of buttons depending on inputs / outputs


PortForwarder
0
1
2
^
v
0

Trane in action



Trane in action



Summary

- OOTM with facilities to combine transformations and experiment, query intermediate results
- Wrapping of different 5 kinds of transformations behind a unique representation, others ?
- The art is in the wrapping, and this is easy and lightweight due to .NET

Future Work

- Type system
  - Explicite/ implicite casting
  - What are types?
- Classification of boxes
- Usability
- Integration of Stratego, and many more boxes

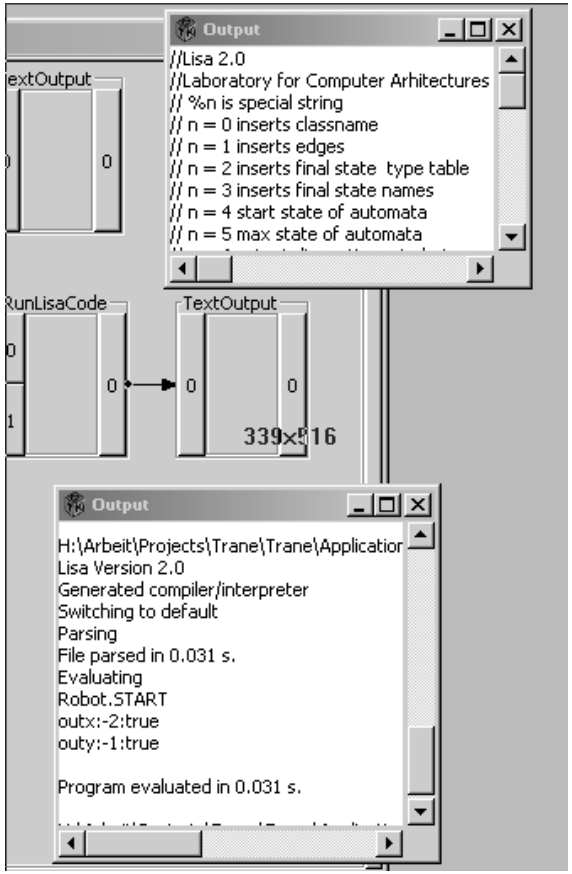
Why does it work?

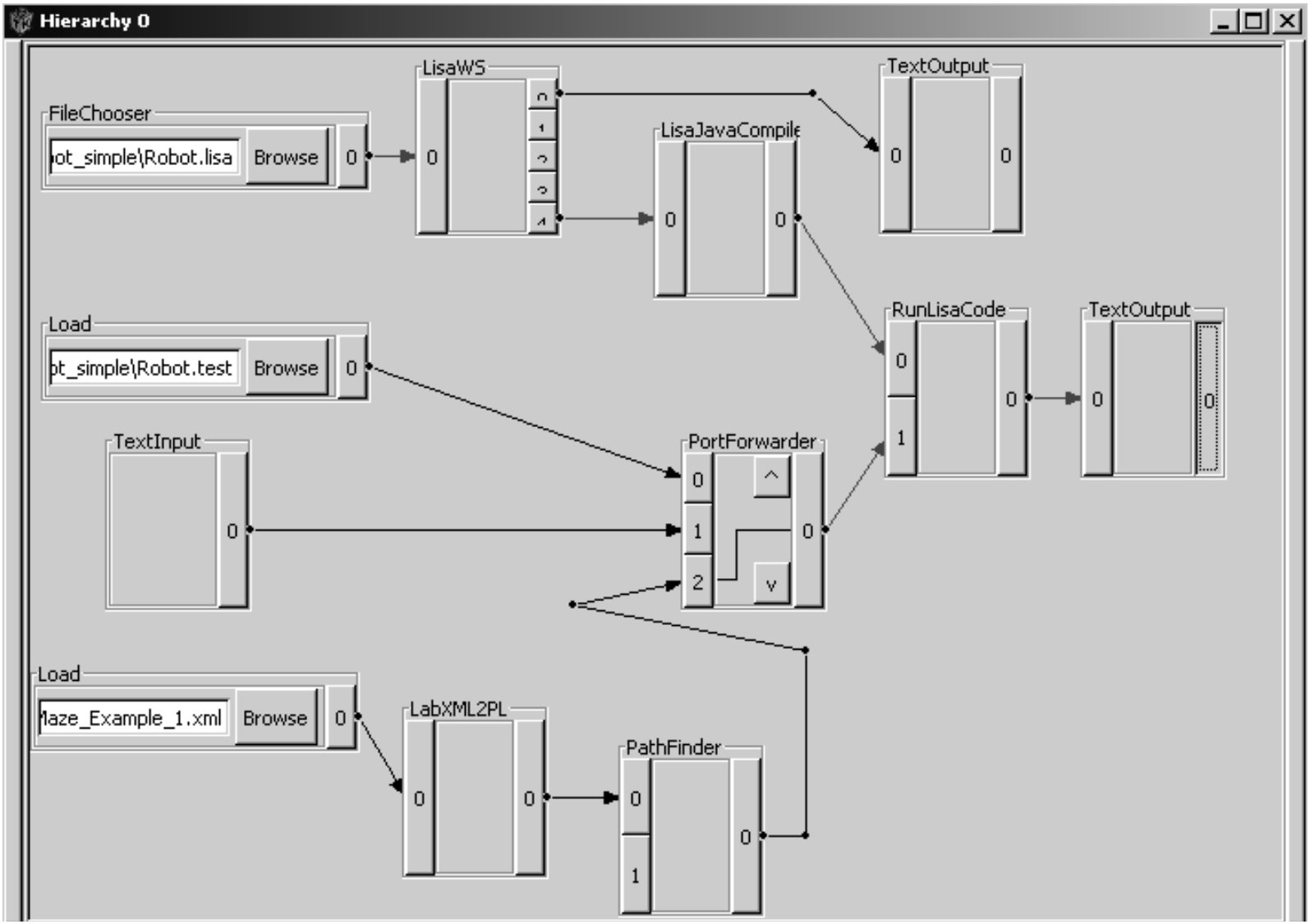
- Every transformation is wrapped in a subclass of *Transformation*, which provides a common interface to interact
- Wrapping is relatively easy due to .NET
- Computation is done using graph traversal
- Values are propagated through converters (also transformations)

Transformation nets

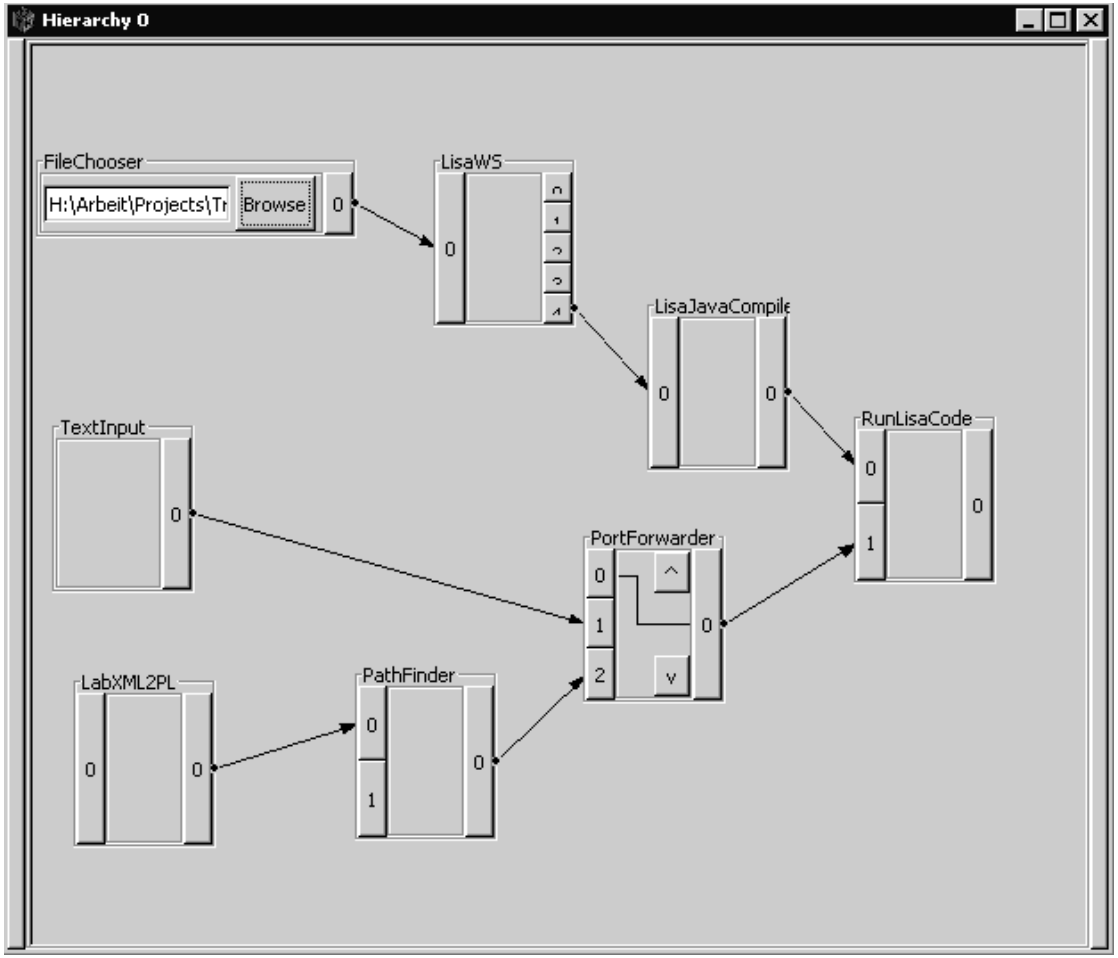
- Boxes have typed input and / or output ports
- Representation is generated or user defined
- Open plugin system: extensible by new transformations at run-time, no configuration files are needed

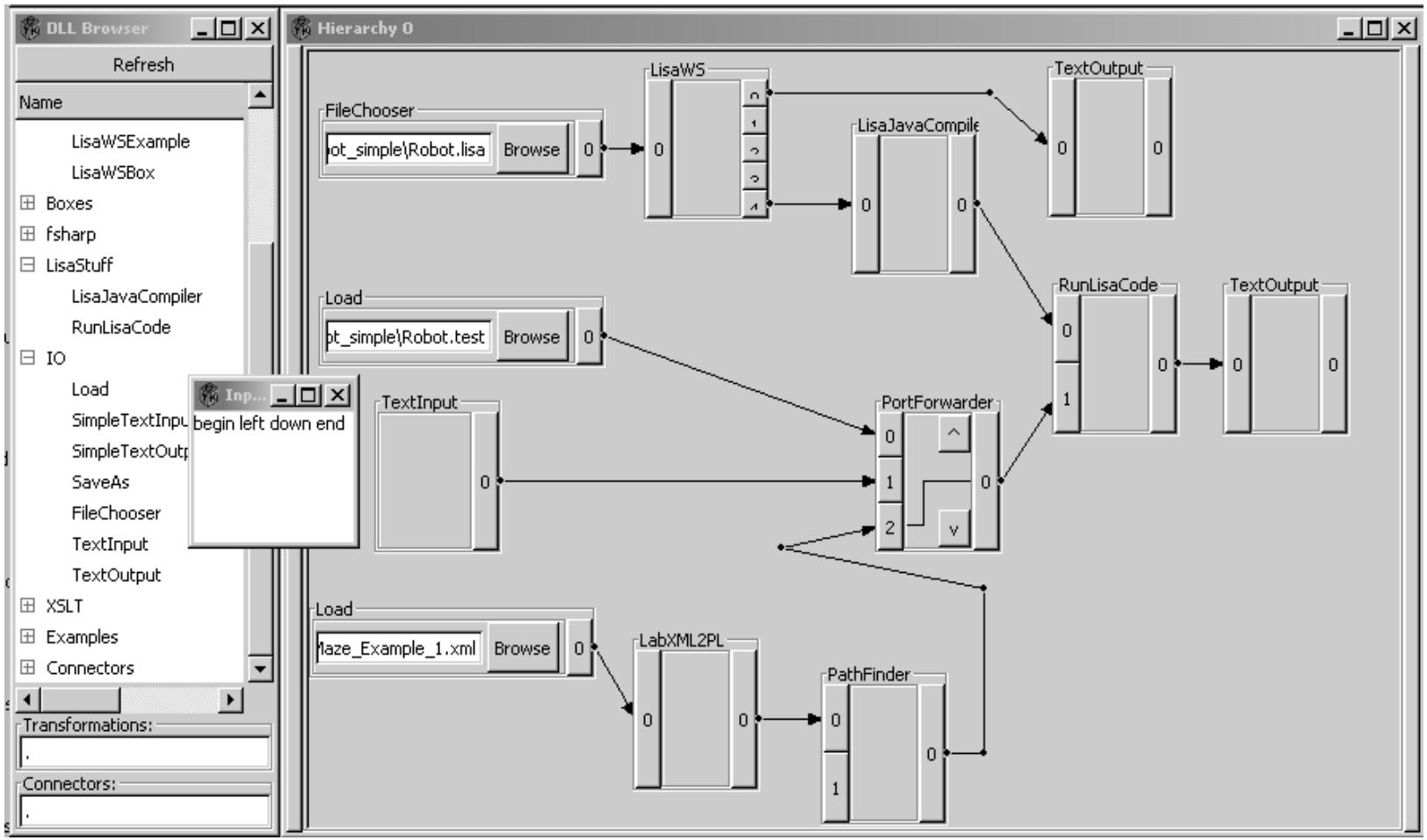
Generated or designed look



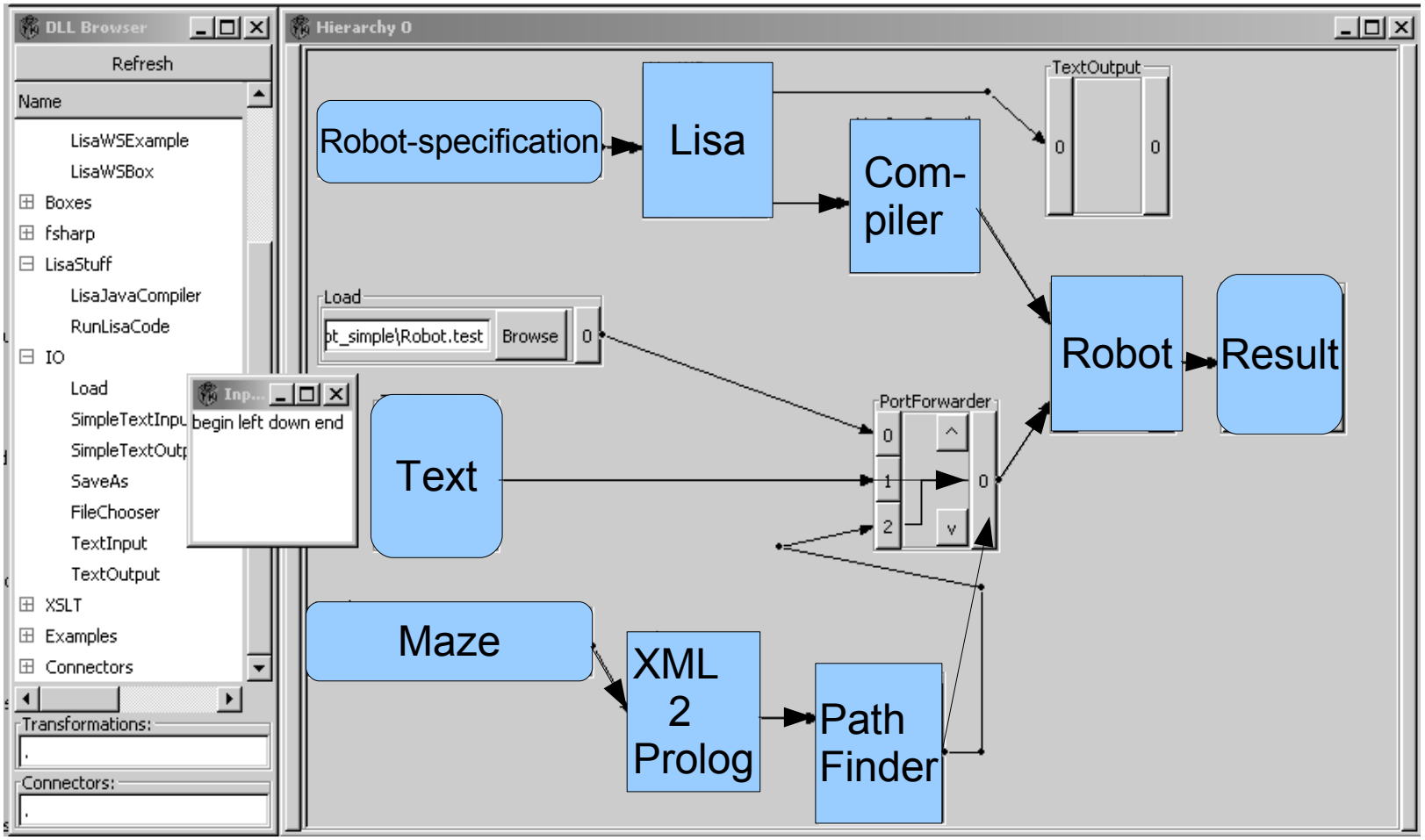

Hierarchy 0
FileChooser
ot_simple\Robot.lisa
Browse
0
LisaWS
0
1
2
3
4
LisaJavaCompile
TextOutput
Load
bt_simple\Robot.test
Browse
RunLisaCode
TextInput
PortForwarder
Load
laze_Example_1.xml
Browse
LabXML2PL
PathFinder

Transformation nets




DLL Browser
Refresh
Name
LisaWSExample
LisaWSBox
Boxes
fsharp
LisaStuff
LisaJavaCompiler
RunLisaCode
IO
Load
SimpleTextInpu
SimpleTextOutp
SaveAs
FileChooser
TextInput
TextOutput
XSLT
Examples
Connectors
Transformations:
Connectors:
Inp...
begin left down end
Hierarchy 0
FileChooser
ot_simple\Robot.lisa
Browse
LisaWS
LisaJavaCompile
TextOutput
Load
pt_simple\Robot.test
Browse
RunLisaCode
TextOutput
TextInput
PortForwarder
Load
Maze_Example_1.xml
Browse
LabXML2PL
PathFinder

What we did...


DLL Browser
Refresh
Name
LisaWSExample
LisaWSBox
Boxes
fsharp
LisaStuff
LisaJavaCompiler
RunLisaCode
IO
Load
SimpleTextInpu
SimpleTextOutp
SaveAs
FileChooser
TextInput
TextOutput
XSLT
Examples
Connectors
Transformations:
Connectors:
Inp...
begin left down end
Hierarchy 0
Robot-specification
Lisa
Com-
piler
TextOutput
0
0
Load
pt_simple\Robot.test
Browse
0
Robot
Result
PortForwarder
0
1
2
0
Text
Maze
XML
2
Prolog
Path
Finder