An Eclipse-based Integrated Environment for Developing Executable Structural Operational Semantics Specifications

Adrian Pop and Peter Fritzson

Programming Environment Laboratory
Department of Computer and Information Science
Linköping University
2006-08-26

SOS'2006, August 26,
Bonn, Germany

Outline

- Introduction
  - Relational Meta-Language (RML)
- Eclipse Environment for RML
  - Framework overview
  - Examples
- Conclusions and Future Work
- Demo

Relational Meta-Language

- a system for generation of efficient executable code from SOS/Natural Semantics specifications
- fast learning curve, used in teaching and specification of languages such as: Java, Modelica, MiniML, Pascal,..
- *developed by Mikael Petterson*
  - *"Compiling Natural Semantics" PhD Linköping University 1996*
  - *also as Springer Lecture Notes in Computer Science (LNCS) vol. 1549 in 1999*
- previously poor environment support (Emacs, command line tools)

SOS/Natural Semantics vs. Relational Meta-Language

RML has the same visual syntax as SOS/Natural Semantics

```
rule    <cond>
        RelName1(H1,T1) => R1 & ...
        RelNameN(Hn,Tn) => Rn &
        ------------------------------
        RelName(H, T) => R
```

RML language properties

- Separation of input and output arguments/results
- Statically strongly typed
- Polymorphic type inference
- Efficient compilation of pattern-matching

SOS/Natural Semantics vs. Relational Meta-Language

Natural Semantics formalism

integers:

$v \in Int$

expressions (abstract syntax):

$$
\begin{aligned}
e \in Exp ::= &\; v \\
&| \; e1 + e2 \\
&| \; e1 - e2 \\
&| \; e1 * e2 \\
&| \; e1 / e2 \\
&| \; -e
\end{aligned}
$$

Relational Meta-Language

```
module exp1:
  (* Abstract syntax of language
   Exp1 *)
  datatype Exp =  INTconst of int
               |  ADDop    of Exp * Exp
               |  SUBop    of Exp * Exp
               |  MULop    of Exp * Exp
               |  DIVop    of Exp * Exp
               |  NEGop    of Exp
  relation eval: Exp => int
end
```

Natural Semantics vs. Relational Meta-Language

Natural Semantics formalism

(1) $v \Rightarrow v$

(2) $$\frac{e1 \Rightarrow v1 \quad e2 \Rightarrow v2 \quad v1+v2 \Rightarrow v3}{e1+e2 \Rightarrow v3}$$

(3) $$\frac{e1 \Rightarrow v1 \quad e2 \Rightarrow v2 \quad v1-v2 \Rightarrow v3}{e1+e2 \Rightarrow v3}$$

(4) $$\frac{e1 \Rightarrow v1 \quad e2 \Rightarrow v2 \quad v1*v2 \Rightarrow v3}{e1+e2 \Rightarrow v3}$$

(5) $$\frac{e1 \Rightarrow v1 \quad e2 \Rightarrow v2 \quad v1/v2 \Rightarrow v3}{e1+e2 \Rightarrow v3}$$

(6) $$\frac{e \Rightarrow v \quad -v \Rightarrow vneg}{-e \Rightarrow vneg}$$

Relational Meta-Language

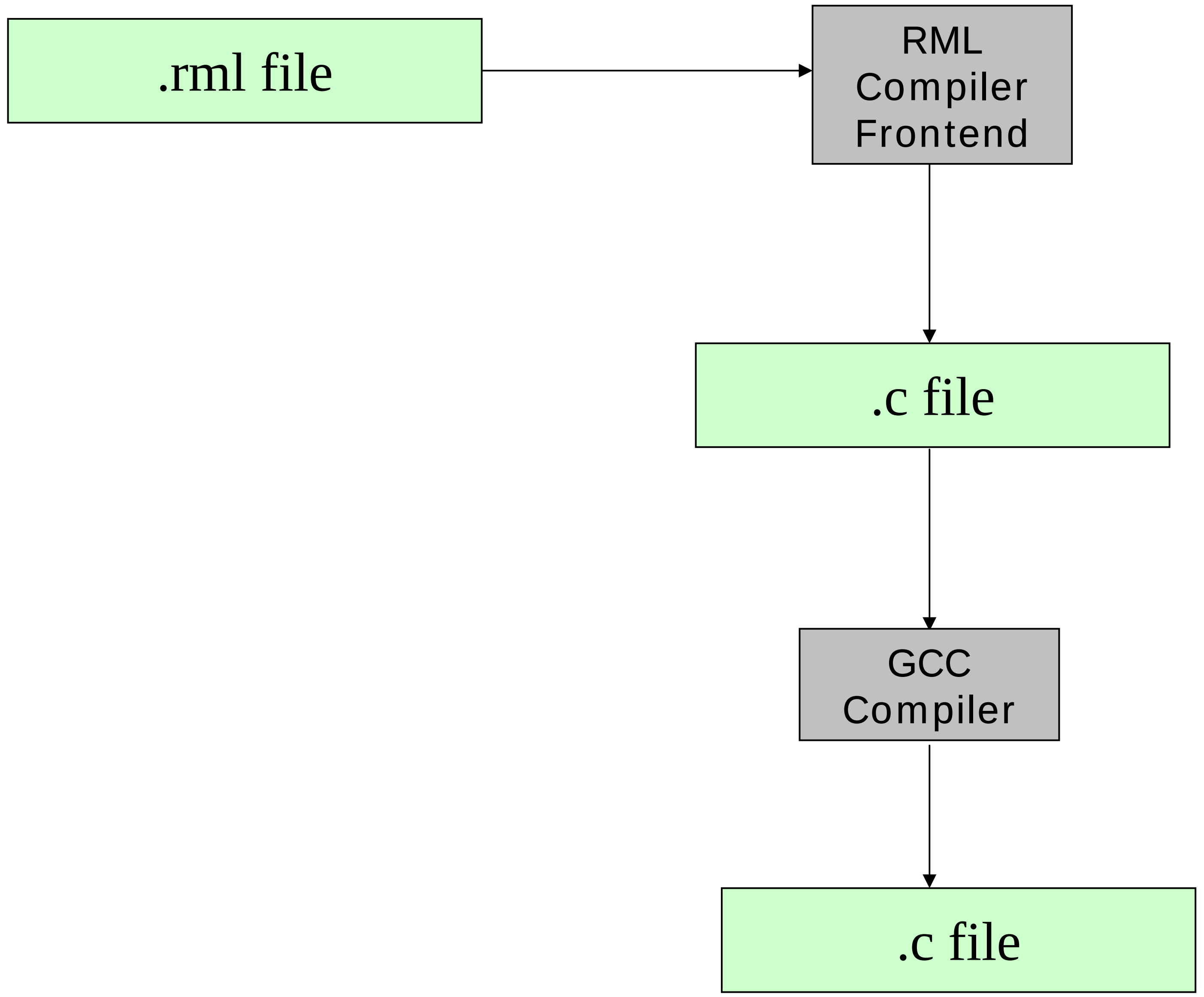
```
relation eval: Exp => int  =

axiom eval(INTconst(ival)) => ival

rule  eval(e1) => v1 &
      eval(e2) => v2 & int_add(v1,v2) => v3
      -------------------------------------
       eval( ADDop(e1, e2) ) => v3

 rule  eval(e1) => v1 &
       eval(e2) => v2 & int_sub(v1,v2) => v3
       -------------------------------------
       eval( SUBop(e1, e2) ) => v3

 rule  eval(e1) => v1 &
       eval(e2) => v2 & int_mul(v1,v2) => v3
       -------------------------------------
       eval( MULop(e1, e2) ) => v3

 rule  eval(e1) => v1 &
       eval(e2) => v2 & int_div(v1,v2) => v3
       -------------------------------------
       eval( DIVop(e1, e2) ) => v3

 rule  eval(e) => v & int_neg(v) => vneg
       ---------------------------------
       eval( NEGop(e) ) => vneg
end (* eval *)
```

The Need for RML Integrated Environment

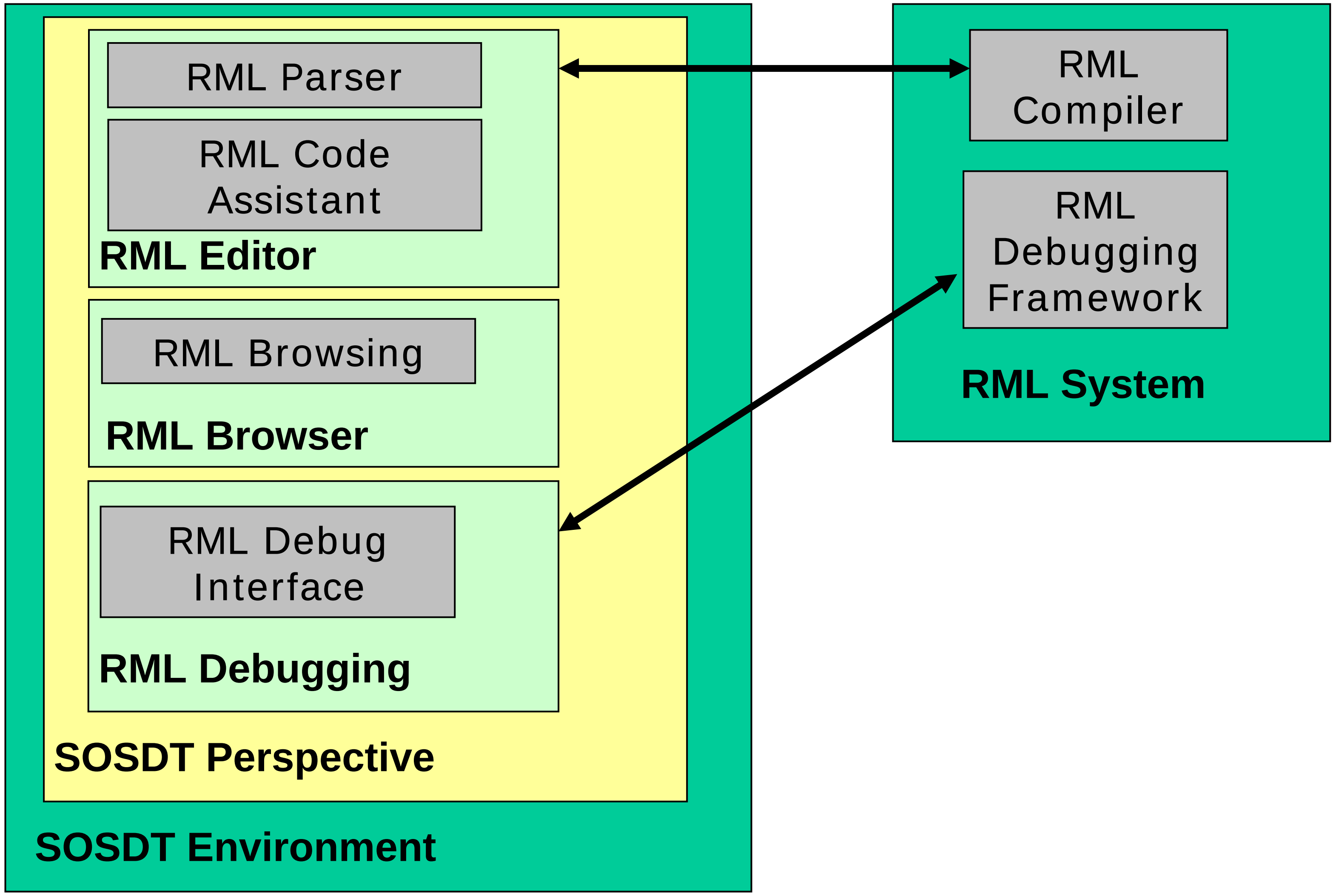
- Facilitate language learning and specification development
  - easy creation of RML projects and modules
  - easy discovery of errors
  - browsing, code highlighting and assistance
  - debugging (step, run, stop and inspect features)
  - code refactorings

- Large specifications are hard to develop
  - Example: The OpenModelica compiler for Modelica
    - **43** packages
    - **57083** lines of code
    - **4054** relations
    - **132** data structures

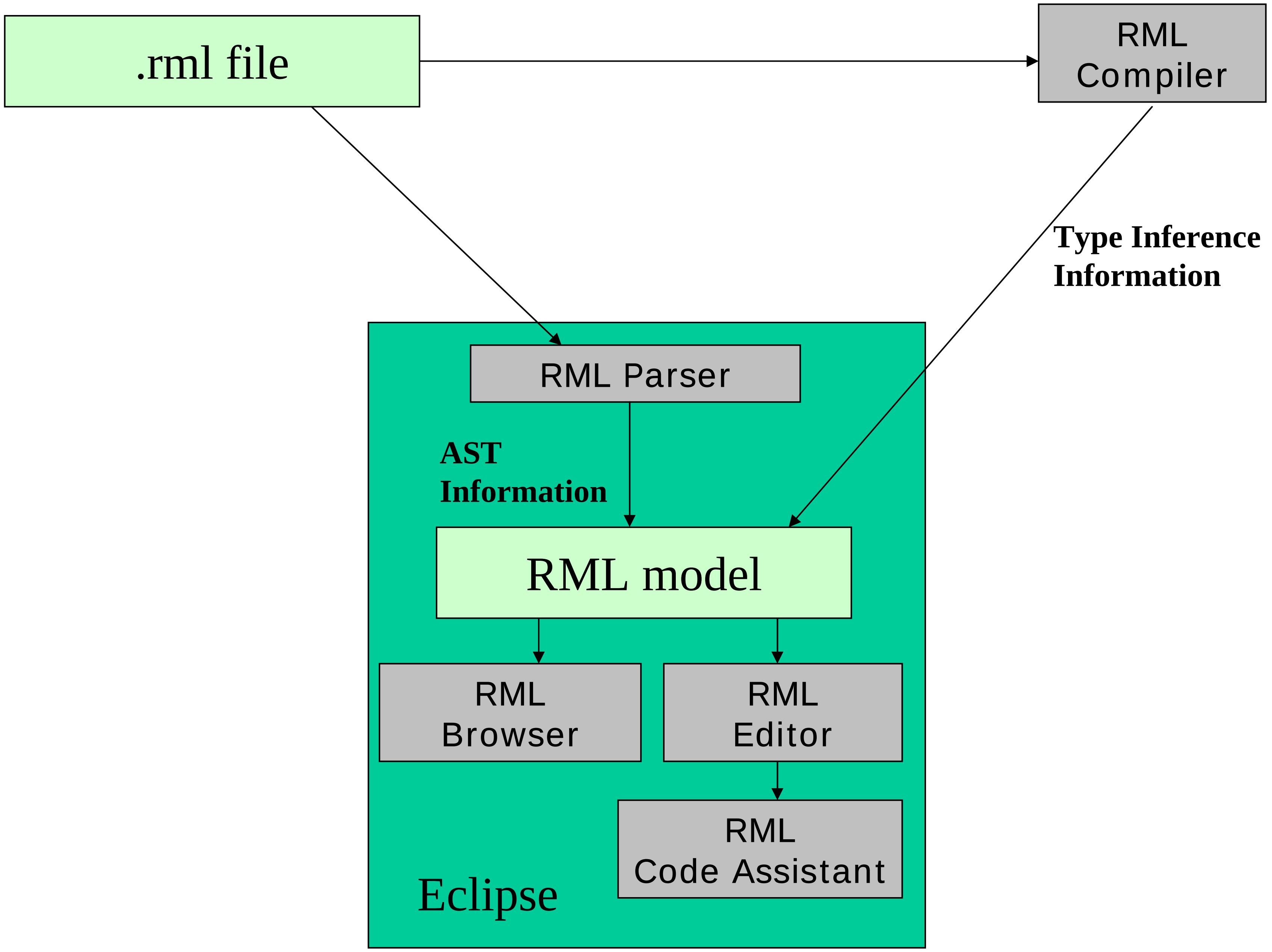
The RML System



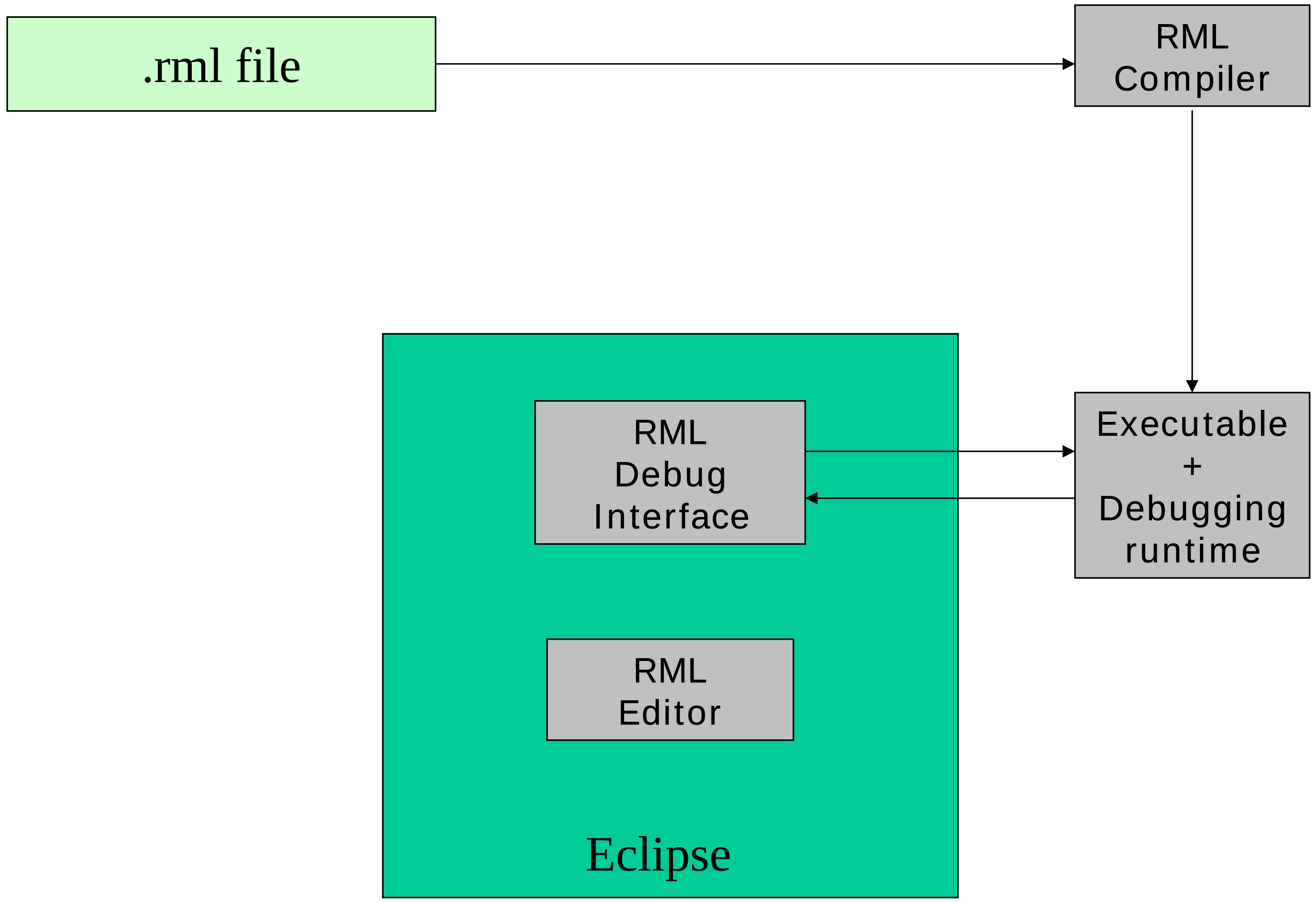
The RML Eclipse Environment – Overview (I)



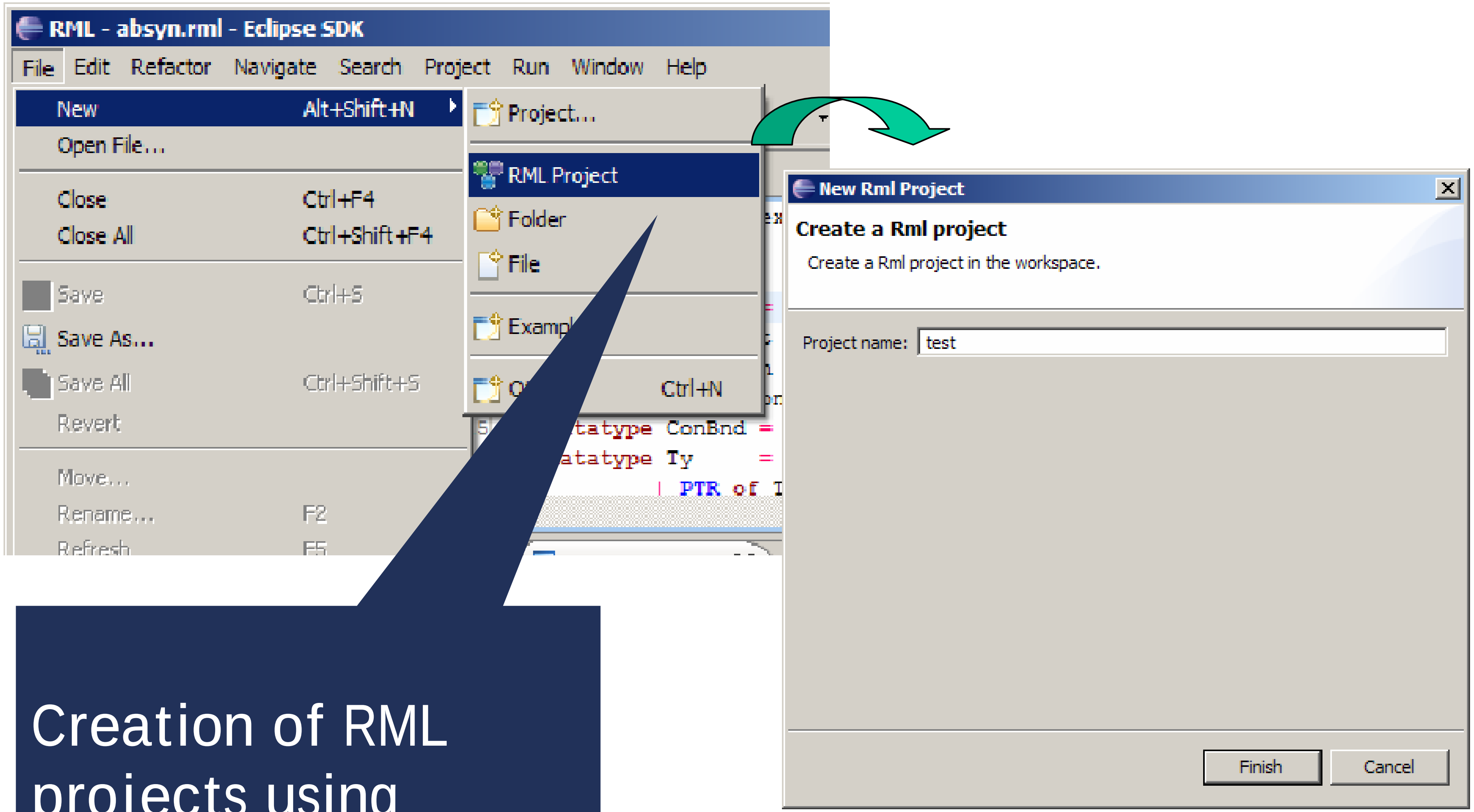
The RML Eclipse Environment – Overview (II)



The RML Eclipse Environment – Overview (III)



Creating RML projects



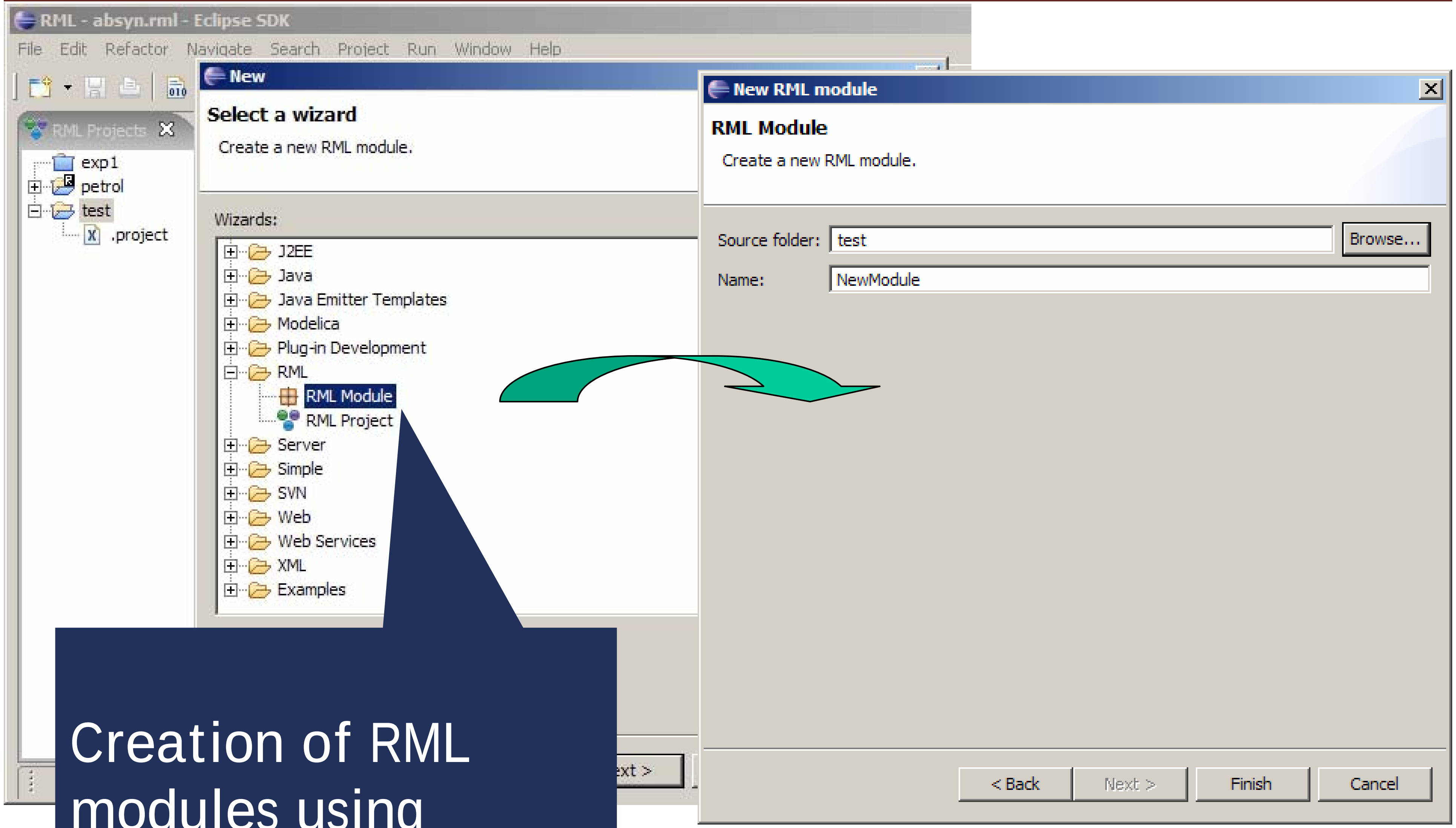
Create a Rml project

Create a Rml project in the workspace.

Project name: test

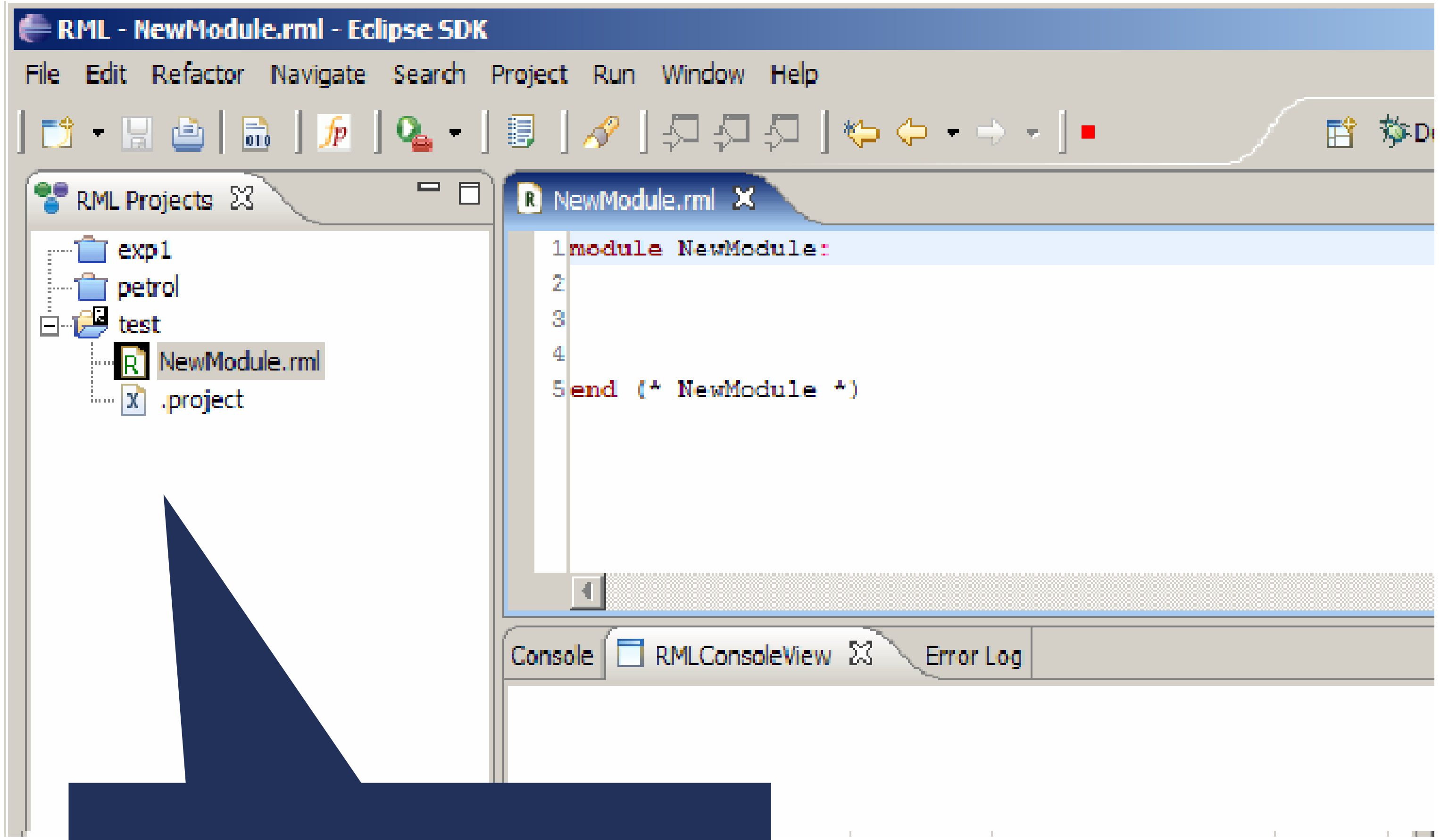
Creation of RML projects using wizards

Creating RML modules (I)



Creation of RML modules using wizards

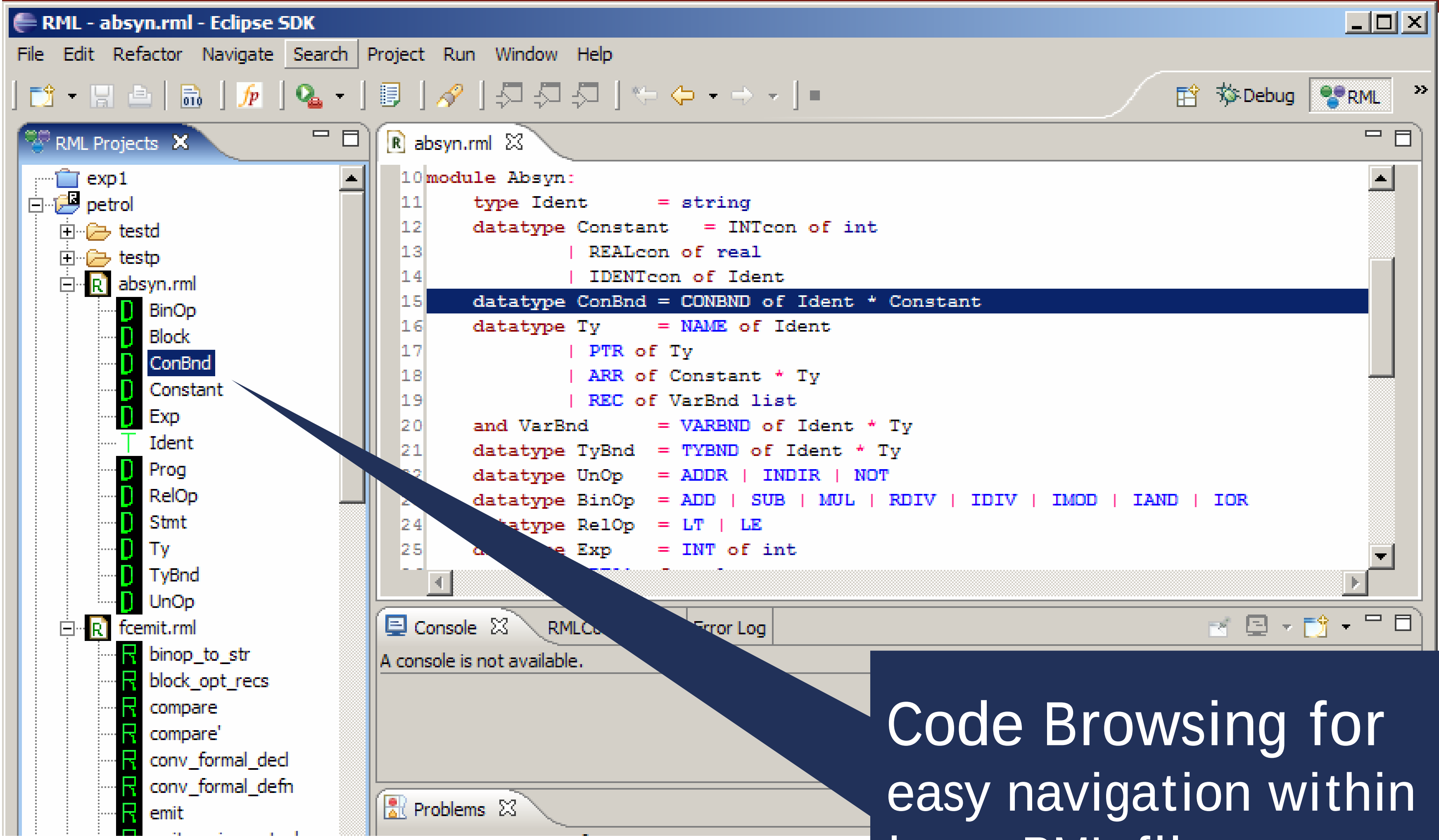
Creating RML modules (II)


RML - NewModule.rml - Eclipse SDK
File Edit Refactor Navigate Search Project Run Window Help
RML Projects
exp1
petrol
test
NewModule.rml
.project
NewModule.rml
Console
RMLConsoleView
Error Log

```
module NewModule:



end (* NewModule *)
```

Creation of RML projects and modules using wizards

Code browsing


RML - absyn.rml - Eclipse SDK

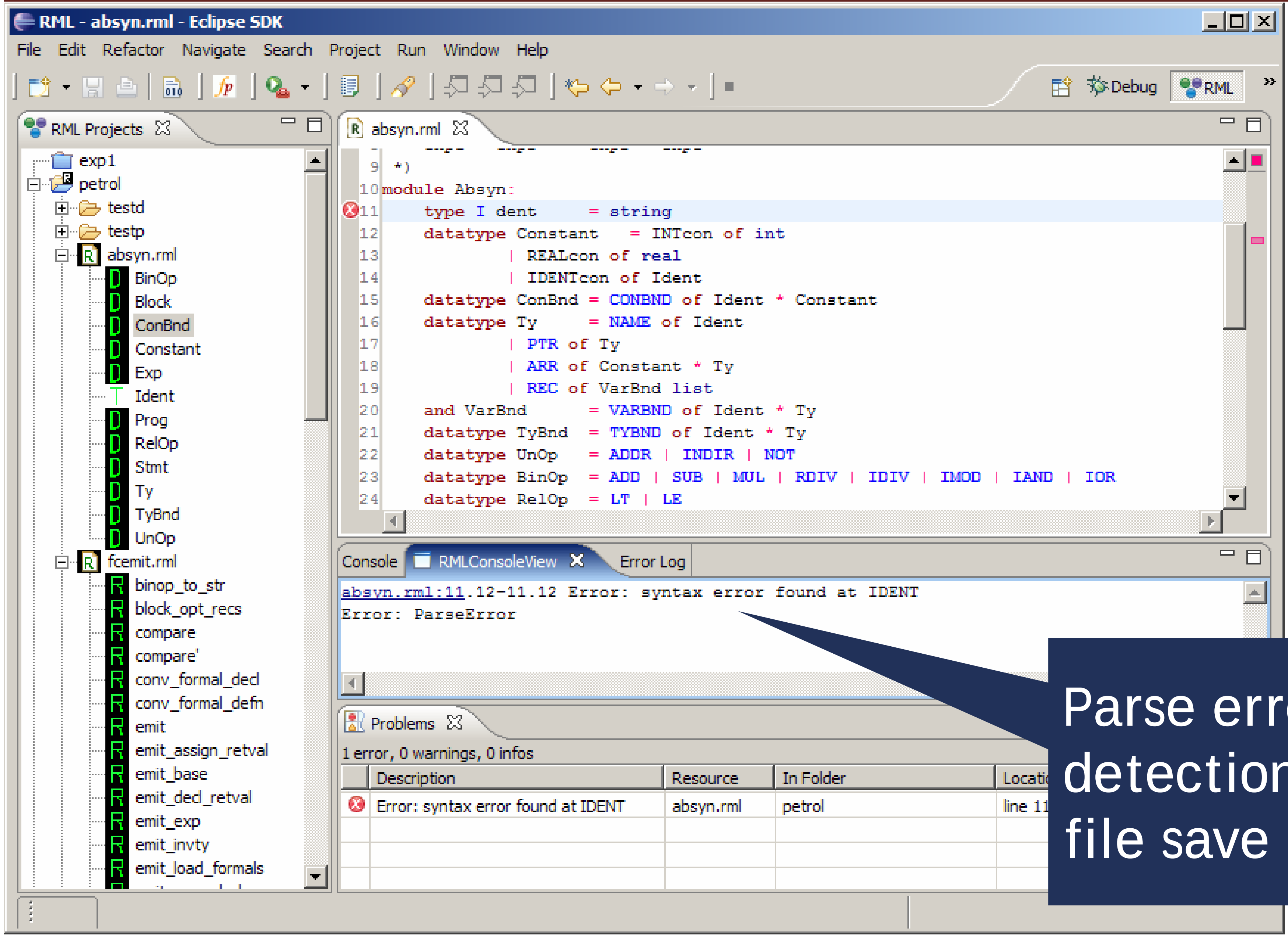
exp1
petrol
testd
testp
absyn.rml
BinOp
Block
ConBnd
Constant
Exp
Ident
Prog
RelOp
Stmt
Ty
TyBnd
UnOp
fcemit.rml
binop_to_str
block_opt_recs
compare
compare'
conv_formal_decl
conv_formal_defn
emit

```
module Absyn:
    type Ident        = string
    datatype Constant    = INTcon of int
              | REALcon of real
              | IDENTcon of Ident
    datatype ConBnd = CONBND of Ident * Constant
    datatype Ty       = NAME of Ident
              | PTR of Ty
              | ARR of Constant * Ty
              | REC of VarBnd list
    and VarBnd        = VARBND of Ident * Ty
    datatype TyBnd    = TYBND of Ident * Ty
    datatype UnOp     = ADDR | INDIR | NOT
    datatype BinOp    = ADD | SUB | MUL | RDIV | IDIV | IMOD | IAND | IOR
    datatype RelOp    = LT | LE
    datatype Exp      = INT of int
```

A console is not available.

Code Browsing for easy navigation within large RML files. Automatic update on file save.

Error detection (I)



```
 *)
module Absyn:
  type I dent      = string
  datatype Constant    = INTcon of int
           | REALcon of real
           | IDENTcon of Ident
  datatype ConBnd = CONBND of Ident * Constant
  datatype Ty      = NAME of Ident
           | PTR of Ty
           | ARR of Constant * Ty
           | REC of VarBnd list
  and VarBnd       = VARBND of Ident * Ty
  datatype TyBnd   = TYBND of Ident * Ty
  datatype UnOp    = ADDR | INDIR | NOT
  datatype BinOp   = ADD | SUB | MUL | RDIV | IDIV | IMOD | IAND | IOR
  datatype RelOp   = LT | LE
```

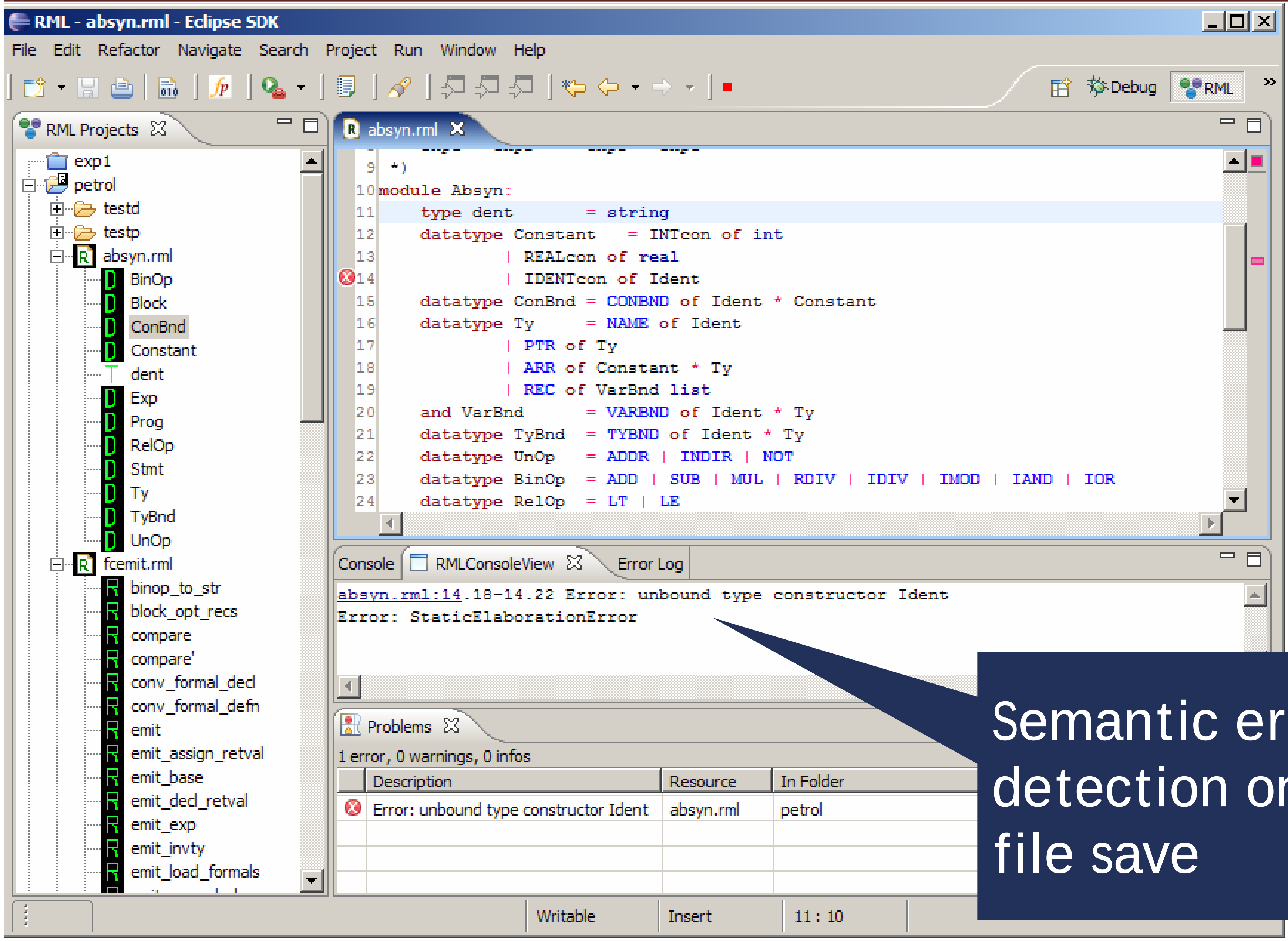
```
absyn.rml:11.12-11.12 Error: syntax error found at IDENT
Error: ParseError
```

1 error, 0 warnings, 0 infos

	Description	Resource	In Folder	Locati
⊗	Error: syntax error found at IDENT	absyn.rml	petrol	line 11

Parse error detection on file save

Error detection (II)



```
 9  *)
10 module Absyn:
11     type dent          = string
12     datatype Constant    = INTcon of int
13              | REALcon of real
14              | IDENTcon of Ident
15     datatype ConBnd = CONBND of Ident * Constant
16     datatype Ty       = NAME of Ident
17              | PTR of Ty
18              | ARR of Constant * Ty
19              | REC of VarBnd list
20     and VarBnd       = VARBND of Ident * Ty
21     datatype TyBnd   = TYBND of Ident * Ty
22     datatype UnOp    = ADDR | INDIR | NOT
23     datatype BinOp   = ADD | SUB | MUL | RDIV | IDIV | IMOD | IAND | IOR
24     datatype RelOp   = LT | LE
```

```
absyn.rml:14.18-14.22 Error: unbound type constructor Ident
Error: StaticElaborationError
```

1 error, 0 warnings, 0 infos

Description	Resource	In Folder
Error: unbound type constructor Ident	absyn.rml	petrol

Semantic error detection on file save

Conclusions and Future work

- Conclusions
  - first prototype of RML Eclipse Environment
  - project and file management
  - code browsing and assistance
  - integrated debugging
- Future Work
  - a lot of bug fixing
  - code folding (comments, relations, etc)
  - refactorings (AST refactorings)
  - better code checking
  - faster debugging
  - more code assistance
  - code templates
  - better integration with the RML compiler

RML Eclipse Environment Demo

Demo

Thank you!
Questions?

RML: http://www.ida.liu.se/~pelab/rml

SOSDT: http://www.ida.liu.se/~adrpo/sosdt