Metamodelling-based ASM Tool-set Development

Alessandro Carioni
Angelo Gargantini
Elvinia Riccobene
Patrizia Scandurra

University of Milan - Italy
University of Bergamo - Italy
University of Milan - Italy
University of Milan – Italy

Pisa Workshop on ASM Open Source Tools, 26-27 January 2007
Outline

- **General motivation of our project**
  - Final goals and meta-modeling approach
  - Metamodel development process

- **The AsmM and its derivatives**
  - Abstract State Machine Metamodel
  - AsmM facilities (XMI/JMI)
  - Concrete textual notation (Asm\(^2\)L)
  - ASM tools interoperability
  - ASMETA toolset

- **AsmM-based simulator**
  - Editor
  - Execution engine
  - Demo
  - Integration with the ATGT tool
State of the Art

- ASM successfully used as a system engineering method
  - able to guide the development of hardware and software systems from requirements capture to their implementation

- The increasing application of ASM for academic and industrial projects caused a rapid development of ASM tools
  - for mechanically verifying properties using theorem provers or model checkers
  - and execution engines for simulation and testing purposes
Analyses of the ASM tools

- Cover only one aspect of the whole system development process
- Have been developed by encoding ASMs into the language of the implementation environment
  - Gofer for AsmGofer, .Net languages for AsmL, C for Xasm, ML for ASM-SL, Promela, SMV and PVS for analysis tools
  - have **syntaxes** strictly depending on the implementation environment
  - adopt **different internal representations** of ASM models
  - provide **proprietary constructs** which extend the basic mathematical concepts of the ASMs
- Do not allow reusing information
Main disadvantages

for a practical and efficient use of ASMs in system development:

1. process of encoding ASM models not always straightforward and natural
   - a practitioner needs to map mathematical concepts (ASM states) into types and structures provided by the target language

2. all ASM tools are loosely coupled and their integration is hard to accomplish
Final goals

We aim:

- To develop a **unified notation**
  - independent from any specific implementation syntax
  - allowing a more **direct encoding** of the ASMs

- To tackle the problem of ASM **tool interoperability**
  - development of a *global framework* for ASM model development and analyses
Metamodelling approach

Proposed by the Model-driven Engineering (MDE)

- an emerging paradigm for software development

Objectives pursued by the MDE:

- separation from system-neutral descriptions and platform-dependent implementations
- identification, specification, separation/combination of specific aspects of a system under development with domain-specific languages (DSLs)
- establishment of precise bridges (or projections) among these different languages in a global framework to automatically execute model transformations

Metamodelling allows to settle this global framework to enable dissimilar languages to be used in an interoperable manner in different technical spaces

- working contexts with a set of associated concepts, knowledge, tools, skills, and facilities
Technical Spaces are similarly organized around a set of concepts. Spaces may be connected via model transformations.

- **Grammarware**
  - Grammar
  - Syntax
  - Program

- **Documentware**
  - Schema
  - XML
  - Document

- **Dataware**
  - Schema
  - DBMS
  - Data

- **Ontology**
  - Top Level O.
  - Ontology engineering

- **MDE**
  - Meta-Model
  - DsLang

Diagram:

- Model transformation and Technical Spaces
- Pisa Workshop on ASM Open Source Tools, 26-27 January 2007
Metamodelling approach

- For language definition too
  - to endow a language or formalism with an abstract notation provided by a *metamodel* (MOF-model)
  - to separate the abstract syntax and semantics of the language from its different concrete notations
  - to allow a XML-based model interchange format
Metamodel development process

1. Requirements capture
2. Metamodel development
3. Metamodel derivatives development
4. Validation of the metamodel and its derivatives

- The process turned out to be iterative
Metamodel development

AsmM comprises:

- an abstract syntax
  - a MOF-compliant metamodel representing in an abstract (and visual) way concepts and constructs of the ASM formalism
- a concrete syntax ($Asm^2L$)
  - an EBNF grammar derived from the AsmM as a textual notation to write ASM models in a textual form
- an interchange syntax
  - a standard XMI-based format automatically derived from the AsmM for the interchange of ASM models

For the metamodel semantics, we adopt the ASM semantics given in Boerger-Staerk
Metamodel development

- metamodel developed in a **modular** way
  - separating the ASM static part (state)
    - domains, functions and terms
  - from the dynamic part (transition system)
  - the ASM rules

- **and bottom-up** way
  - Basic ASMs, Turbo ASMs and Multi-Agent (Sync/Async) ASMs
Metamodel development
Metamodel derivatives development

- An XMI interchange format
- Java Metadata Interfaces to access and manipulate models in a MOF repository
- A concrete syntax (Asm²L) together with a parser for typechecking and MOF model storing
Validation of AsmM

- Validate the expressive power of Asm^2L w.r.t. ASM models
  - test if Asm^2L is suitable to encode not trivial ASM specs
  - check if the encoding process of mathematical models is natural and straightforward
- Validate that Asm^2L can match up all existing ASM tools' languages
  - test that Asm^2L is suitable to encode ASM specs given in ASM dialects
Validation of AsmM derivatives

- Check if the AsmM-specific XMI and JMIs provide the desired global infrastructure for interoperability and integration of ASM tools
  - developing an ASM simulator

- Test if the metamodelling approach is applicable to existing tools and evaluate the effort it requires in modifying existing code
  - we make ATGT (an existing tool for ASM test case generation -- written in AsmGofer syntax) AsmM-compliant
10 reasons to metamodel ASMs

1. To have a **standard abstract notation** as unified representation of ASM concepts independent of any particular implementation platform
2. To have a **graphical representation** of the ASMs also useful for teaching purposes
3. To have an **interchange format** among ASM tools
4. To have **standard libraries** and APIs to use in new or existing tools and programs
5. To automatically derive a **family of languages**, visual/textual notations and their parsers
10 reasons to metamodelf ASMs (2)

6. To allow tool interoperability
7. To help the integration of existing tools
8. To help the development of new tools
9. To export ASMs to other environments and permit the integration with other specialized external notations and tools
10. To complement the MDE with a formal approach
Outline

- General motivation of our project
  - Final goals and metamodelling approach
  - Metamodel development process

- The AsmM and its derivatives
  - Abstract State Machine Metamodel
  - AsmM facilities (XMI/JMI)
  - Concrete textual notation (Asm²L)
  - ASM tools interoperability
  - ASMETA toolset

- AsmM-based simulator
  - Editor
  - Execution engine
  - Integration with the ATGT tool
The OMG’s Metamodelling Framework

conformsTo Ξ instanceOf

A OT-based framework for MDE!
The **Meta-Object Facility**

- **Classes** with attributes and operations, possibly inherited from other classes by **Generalization**

- **Associations** between classes (simple, composite, shared aggregation) with cardinality and uniqueness

- **Packages** to group elements for modularizing

- **Data types** whose values do not have object identity
  - primitive types: Boolean, Integer, and String
  - data type constructors: Enumeration, Structure, Collection, etc.

- **Constraints** (**well-formedness rules**) in the **Object Constraint Language**

Pisa Workshop on ASM Open Source Tools, 26-27 January 2007
AsmM Architecture

(about 115 classes, 114 associations, 150 OCL rules)

The structural language

The definitional language

The language of rules

The language of terms

Pisa Workshop on ASM Open Source Tools, 26-27 January 2007
OCL constraint:

context Asm
inv A1: mainrule->notEmpty() implies
mainrule.arity=0

The structural language - **Backbone**
Asm Metamodel

The structural language - Header
Asm Metamodel

The structural language - Body
**Asm Metamodel**

The language of terms – *Basic Terms*

Pisa Workshop on ASM Open Source Tools, 26-27 January 2007

**OCL constraint:**
context FunctionTerm
ingv T8: domain = function.codomain
The language of rules – *Basic Transition Rules (a fragment)*

OCL constraint:
context UpdateRule
inv R2: (location.oclIsTypeOf(LocationTerm) and not location.oclAsType(LocationTerm).function.oclIsTypeOf(Monitored)) or (location.oclIsTypeOf(VariableTerm) and location.oclAsType(VariableTerm).kind = VariableKind::locationVar )
Asm Metamodel

The language of rules – *Turbo Transition Rules (a fragment)*

And so on…
Derivative artifacts from the AsmM

I
MOFtoJava
MOFtoXML
MOFtoEBNF

II
AsmM JMI
AsmM XMI
Asm\textsuperscript{2}L

III
conformsTo
conformsTo
conformsTo

ASM model
ASM model
ASM model

Java objects
XMI file
textual spec.

M0
M1
M2
M3

data
ASM model
MOF

Modelware
Java env.
XMLware
grammarware

Technical Spaces

Pisa Workshop on ASM Open Source Tools, 26-27 January 2007
Deriving **human-usable textual notations from metamodels**

- Deriving textual concrete notations from a MOF-compliant metamodel - **forward engineering** - is not yet well established!

- Complex MOF-to-text tools exist
  - HUTN, Anti-Yacc, MOF2Text, JET for EMF, etc.
  - capable of generating text (grammars) from specific MOF-based repositories
  - either they do not allow a **detailed customization of the generated language** or they are only one way: from MOF to text

- The “opposite” process
  - from EBNF grammars to MOF - **reverse engineering** – has been more intensively studied
Asm\textsuperscript{2}L: our experience on Bridging Modelware and Grammarware

- We define general rules to derive a context-free EBNF grammar from a MOF-compliant metamodel (MOF v. 1.4)
- We use these mapping rules to instruct a JavaCC parser generator to get a compiler capable of
  - parsing the grammar and
  - transferring information about models into a MOF-based instance repository (text to MOF objects)
- We apply the proposed approach to the Abstract State Machine metamodel (AsmM)
**Rules from MOF to EBNF/JavaCC**

- Translation for the following MOF (1.4) elements to EBNF/JavaCC
  1. Class
  2. Data Type (primitive and constructors)
  3. Attribute (boolean, String, …)
  4. References (Association and Association End)
  5. Generalization
  6. (OCL) Constraints
A class $C \rightarrow$ a non terminal symbol $C$. User-defined keywords delimit the expression in the derivation rule for $C$. The expression represents the actual content of $C$ and is determined by the full descriptor (for attributes and associations) of the class according to the other rules.

The start symbol is the non-terminal corresponding to the root class.

MOF

<table>
<thead>
<tr>
<th>Asm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

EBNF

Asm ::= "asm" ...

Pisa Workshop on ASM Open Source Tools, 26-27 January 2007
Class - JavaCC

- The **Asm²L compiler** (about **6852** lines of code)
- We instruct **JavaCC**, a traditional **Parser Generator** for grammars, to instantiate a MOF-based repository through **JMI**s
- For each class C, in JavaCC one method C() parses the grammar symbol C and returns a JMI instance of the class C

```java
C C(): {
   // create result a new instance
   // temp variables for attributes
}
// expansion unit
<startC> // expression starting delimiter
// read content of C and fill the instance
<endC> // expression ending delimiter
}
```

```java
Asm Asm(): {
   Asm result = AsmStructure.getAsm().createAsm();
   // temp variables for attributes and references
}
// expansion unit
"asm" // start Asm
// read content of Asm and fill the instance result
<EOF> // end Asm
}```
OCL Constraints

- OCL constraints are **not** mapped to EBNF concepts

- Appropriate **actions are added to the JavaCC code** to check whether the input model is well-formed or not

- In our case, we explicitly implemented in Java an OCL checker
  - by hard-encoding the OCL rules of AsmM (one method checkC for OCL constraints of class C)
  - constraint incompatibility errors are detected and reported using standard **Java exception handling**

- Alternatively, an OCL compiler could be connected to the generated parser for the constraint consistency check
A Flip-Flop Device in Asm²L

Macro FSM

Main rule

\[
\text{FSM}(i, \text{cond}, \text{rule}, j) = \\
\begin{cases} 
  \text{if } \text{ctl}_\text{state} = i \text{ and } \text{cond} \\
  \text{then } \{ \text{rule}, \text{ctl}_\text{state} := j \} \\
  \text{endif} \\
\end{cases}
\]

\[
\text{FLIPFLOP} = \{ \text{FSM}(0, \text{high}, \text{skip}, 1), \text{FSM}(1, \text{low}, \text{skip}, 0) \}
\]

rule \ r_Fsm($i$ in State, $\text{cond}$ in Boolean, $\text{rule}$ in Rule, $j$ in State) =
  \text{if } \text{ctl}_\text{state} = i \text{ and } \text{cond} \\
  \text{then } \text{par } \text{rule} \\
  \quad \text{ctl}_\text{state} := j \\
  \text{endpar} \\
  \text{endif}

main rule \ r\_flip\_flop = \text{par} \\
  r_Fsm(0, \text{high}, <<\text{skip>>}, 1) \\
  r_Fsm(1, \text{low}, <<\text{skip>>}, 0) \\
  \text{endpar}

Taken from:

A Flip-Flop Device in Asm²L

Header section
Import/export clauses, ASM signature

Import
asm flip_flop
import StandardLibrary

Signature:
domain State subsetof Natural
dynamic controlled ctl_state : State
dynamic monitored high : Boolean
dynamic monitored low : Boolean

Initialization section
Dynamic domains and functions initializations

default init initial_state:
function ctl_state = 0
function high = false
function low = false

Body section
Static domains/functions defs., rule decls., and axioms

Definitions:
domain State = {0,1}
rule r_Fsm($i in State, $cond in Boolean,$rule in Rule, $j in State) =
    if ctl_state=$i and $cond
        then par $rule
            $ctl_state := $j
        endpar
    endif

Main rule
Main rule r_flip_flop = par
    r_Fsm(0,high,<<skip>>,1)
    r_Fsm(1,low,<<skip>>,0)
endpar
Dining Philosophers in Asm²L


agent program

```
DININGPHILOSOPHER =
    if owner(resource) = none then owner(resource) := self
    if owner(resource) = self then owner(resource) := none

formally resource(self) = (leftFork(self), rightFork(self))
```

**rule** r_DiningPhilosopher($self in Philosophers) = par r_Eat[$self] r_Think[$self] endpar

**macro rule** r_Think($self in Philosophers) =
    if owner(left_fork($self) = $self and owner(right_fork($self)) = $self
    then par
        owner(left_fork($self)) = none
        owner(left_fork($self)) = none
    endpar endif

**macro rule** r_Eat($self in Philosophers) =
    if owner(left_fork($self) = none and owner(right_fork($self)) = none
    then par
        owner(left_fork($self)) = $self
        owner(left_fork($self)) = $self
    endpar endif
ASMETA tool set 1/2
http://asmeta.sourceforge.net/index.html
complete Java-based code under GPL licence

- **The AsmM Abstract Syntax** (the metamodel)
  - edited with the Poseidon tool v.2.6 according to the UML profile for MOF

- **The AsmM Concrete Syntax (Asm²L)**
  - The EBNF grammar developed by means of the Sun/JavaCC
  - A quick guide and Examples of ASM specs (ZIP archive)

- **The AsmM XMI Format** (conforms to MOF 1.4, XMI 1.2)
  - AsmM XMI DTD generated by the MOF-MDR for NetBeans
  - AsmM JMI interfaces generated by the MOF-MDR for NetBeans

- **The AsmM Standard Library**
  - StandardLibrary.asm for predefined ASM domains and functions
ASMETA tool set

- A graphical front-end called **ASMEE** (ASM Eclipse Environment)
  - An IBM-**Eclipse plug-in** to edit and manipulate AsmM models

- The **AsmM-Parser** to
  - process the **Asm²L specs**
  - check for their consistency w.r.t. the **OCL constraints**
  - generate their equivalent **XMI representation**
  - translate them into **AsmM instances** in a **MOF-MDR for NetBeans** repository by using the AsmM JMI's

- The **AsmM-Simulator**
  - to make AsmM models executable
  - an **interpreter** which navigates through the **MOF-MDR for NetBeans** repository, where ASM models are instantiated

- The **ASM Tests Generation Tool (ATGT)**
  - Now, AsmM-compliant
Outline

- **General motivation of our project**
  - Final goals and metamodelling approach
  - Metamodel development process

- **The AsmM and its derivatives**
  - Abstract State Machine Metamodel
  - AsmM facilities (XMI/JMI)
  - Concrete textual notation (Asm²L)
  - ASM tools interoperability
  - ASMETA toolset

- **AsmM-based simulator**
  - Editor
  - Execution engine
  - Integration with the ATGT tool
Asmeta toolset

- Every tool will have its eclipse counterpart
  - parser → asm2l editor plugin
  - interpreter → executor as plugin eclipse
- Every tool will work at command line:
  - parser:
    ```
    java -jar asm_parser.jar <filename>
    ```
  - interpreter
    ```
    java -jar asm_interpreter.jar <filename>
    ```
    ...
- parser and interpreter do not depend on eclipse
  - the eclipse plugin just calls the parser and interpreter
Asm²L Editor

- plugin for eclipse
- multiplatform
- provides:
  - from eclipse
    - project, editor fonts …
    - automatic update
    - help system
    - …
  - DEMO …

for Asm2L
- syntax highlighting
- new file wizard
- preferences
- …
The Asmeta stack

- Abstract State Machines Theory
- ECLIPSE PDE
- MOF
- JMI
- JavaCC
- Asm Interpreter
- Asm2L: a language for the Asm
- ASMEE Simulator
- ASMEE Editor
- AsmM: Asm Metamodel
license information

- asmeta is released with a double licensing scheme
  - **GPL**: *free* to use and modify
    - if you modify and distribute the modified version you need to include the source code and keep the original license (still GPL)
    - derivative work must be GPL compatible: you cannot include asmeta in a proprietary package
  - you can use asmeta to write proprietary specifications: the license does not apply to the files you use with asmeta
    - **ALSO FOR COMMERCIAL** (not like ASML “You may not use or distribute this Software or any derivative works in any form for commercial purposes.”)
  - modified to allow the use of log4j
  - **proprietary license**: if you believe the GPL is too strict, we can provide asmeta with another license
Why open source

- **developers:**
  - can contribute modifying the code
  - their code remains “public”
  - still, can take credit and sell their expertise
  - **in the ASM community:** can write their addon and tools

- **user**
  - no limitation in the use of the software
    - no hidden cost, no hidden clauses, …
  - a community (and not a company) is behind the software
  - can see and judge also the technology behind the software
  - academic user or student: can understand better the ASMs looking as they are implemented
Development notes

- Hosted at sourceforge.net
  - asmeta.sf.net
  - already working:
    - web site, bug trackers, svn for source code
  - in the future:
    - mailing lists, forums, release system, news
- ROADMAP … to decide after this meeting – probably for ASM2007 first stable release

- log4j for logging
  - every user can personalize its log (for debugging of the asm specs and of the interpreter)

- JUnit and Fit (http://fit.c2.com/) for unit testing

- emma to evaluate coverage
Parser and Interpreter

- Parser and Interpreter work together
- The interpreter works on ASM instances of the repository
  - access to MOF object by JMI
  - ignore how the ASMs and their parts are created

![Diagram of parser, repository, and interpreter](image)
The ASM interpreter

1. architecture
   - java and reflective visitors
2. supported constructs
   - briefly: all except turbo
3. extra feature
   - axiom checking
   - inconsistent updates detection
   - logging
Main Classes

- **Value**: represents the possible values of ASM terms
  - it’s a hierarchy: IntegerValue, SetValue, …
  - represented in corresponded Java Object whenever possible
- **State**: map from Terms to Values
  - ex.: let a be an IntegerTerm, currentState(a) := 10
- **Assignements**: map from VariableTerms to Values
- **UpdateSet**: a map between location to values
  - operation of composition
  - check if two UpdateSets are consistent
- **Environment**: values for monitored functions
The interpreter at work

- current STATE
- Assignement
- Rules
- Env

INTERPRETER

- UpdateSet

next STATE

the interpreter visits the current state and the rules

Pisa Workshop on ASM Open Source Tools, 26-27 January 2007
Visiting Terms and Rules

- **TermVisitor**: for every subclass of `Term`, it introduces a method `visit` which returns the Value for object of that subclass
  - for example for conditional terms:
    ```java
    public Value visit(ConditionalTerm condTerm)
    ```

- **RuleVisitor**: for every subclass of `Rule`, it introduces a method `visit` that builds the update set for rules of that subclass
  - for example
    ```java
    public UpdateSet visit(UpdateRule rule)
    ```

- The dispatching is done dynamically (using reflection)
  - Terms and Rules do not need a visit method
  - the discovery is dynamic and not all the methods visit must be implemented
Domain supported

- Integers
- Reals
- Booleans
- String
- Undef
- Enumerations
- Sets
- Sequences
- Tuples

and terms of these types
Further terms

- **Conditional term**
  - `if φ then t else s endif`

- **Switch-case term**
  - `switch t case t1:s1…case tn:sn otherwise Sn+1 endswitch`

- **Collection terms**
  - `set`
    - `{t1,…,tn}`
  - `sequences`
    - `[t1,…,tn]`
  - `Set comprehension`
    - `{t | x1 in D1…xn in Dn with φ}`
Example of a visit method

```java
public Value visit(ConditionalTerm condTerm) {
    BooleanValue guard =
        (BooleanValue) visit(condTerm.getGuard());
    if (guard.getValue()) {
        return visit(condTerm.getThenTerm());
    } else if (condTerm.getElseTerm() != null) {
        return visit(condTerm.getElseTerm());
    } else {
        return UndefValue.UNDEF;
    }
}
```
Quantification terms

- existential term
  - $\text{exist } x_1 \text{ in } D_1 \ldots x_n \text{ in } D_n \text{ with } \varphi$

- universal
  - $\text{forall } x_1 \text{ in } D_1 \ldots x_n \text{ in } D_n \text{ with } \varphi$

- let
  - $\text{let } x_1 = t_1, \ldots, x_n = t_n \text{ in } t \text{ endlet}$
  - example
    // $a*x^2+b*x+c=0$
    let $\delta = \sqrt{b*b - 4*a*c} \text{ in }$
    $((-b + \delta) / 2*a, (-b - \delta) / 2*a)$
    endlet
Standard Library

- arithmetical operators
  - +, -, *, /, %
- relational operators
  - =, !=, <, >, <=, >=
- boolean operators
  - not, and, or, implies, iff
- functions for sets and sequences
  - first(), tail(), union()…

- still evolving – not really standard yet
- the interpretation of standard function is done using reflection:
  - for a function $f$ search a method $f$
Rules

- Update rule
  - \( f(t_1, \ldots, t_n) := t \)

- parallel
  - \( \text{par } P_1 \ldots P_n \text{ endpar} \)

- sequential composition
  - \( \text{seq } P_1 \ldots P_n \text{ endseq} \)

- extend
  - \( \text{extend } D \text{ with } x_1, \ldots, x_n \text{ do } P \)

- let
  - \( \text{let } x_1 = t_1, \ldots, x_n = t_n \text{ in } P \text{ endlet} \)
Selection rules

- conditional
  - if $\varphi$ then $P$ else $Q$ endif

- case
  - switch $t$ case $t_1$: $P_1$…case $t_n$: $P_n$ otherwise $P_{n+1}$ endswitch

- non deterministic choose
  - choose $x_1$ in $D_1$,…,$x_n$ in $D_n$ with $\varphi$ do $P$ ifnone $Q$

- parallel execution for all
  - forall $x_1$ in $D_1$,…,$x_n$ in $D_n$ with $\varphi$ do $P$
macro rules

- macro rules can have parameters
  - definition: \( r(x_1 \text{ in } D_1, \ldots, x_n \text{ in } D_n) = P \)
  - call: \( r[t_1, \ldots, t_n] \)
  - semantics: \( P[t_1/x_1, \ldots, t_n/x_n] \) (substitution)

- introduction of auxiliary classes like `RuleSubstitution` using a reflective visitor which visits a rule and returns a new rule with the right substitution

- **Substitution requires renaming:**

  \[
  (\text{let } x = 1 \text{ in } f := x + y)[x/y] \]

  \[
  \text{let } x = 1 \text{ in } f := x + x \quad \text{Error}
  \]

  \[
  \text{let } z = 1 \text{ in } f := z + x \quad \text{Ok!}
  \]
DEMO .....
Function and Basic Terms

- Natural, Integer, Strings,…
- Static functions
  - standard arithmetic functions and functions over collections in the StandardLibrary
  - defined by the user
    - function \( f(x_1 \text{ in } D_1, \ldots, x_n \text{ in } D_n) = t \)
  - can be recursive
- **QuickSort**
  - *given a disordered array, order it (using a function)*
  - static functions: disordered
  - exists term
  - recursive functions like qsort
Macro rules

- A rule can call other rules
  - subprograms
- **AdvancedClock1**
  - an advanced clock that at every step increments the second (and if necessary the minutes and hours) taken from AsmBook
  - domain Second subsetof Integer
  - domain Second=[0..59]
  - macro rule r_IncMinHours = par
  - main rule r_AdvancedClock = par
    - if seconds = 59 then r_IncMinHours[] endif
    - seconds := (seconds+1) mod 60 endpar
Macro rules

- A macro can contain parameters
- **Swap**

```plaintext
macro rule r_swap ($x in Integer,
                    $y in Integer) =

par
$x := $y
$y := $x
endpar
```

_copy semantics_
Monitored quantities

- Monitored functions are updated by the environment
- The environment can be
  - a file
  - the standard input
- **AdvancedClock2**
  - *like AdvancedClock but a signal that increments the seconds*
Choose

- the interpreter supports non deterministic choice among finite domains
  - the non determinism is pseudo random
    - but from an execution to another it changes

- **swapSort**
  - *take to elements not in order and swap them*
    - *until the array is ordered - the update set is empty*
  - macro with parameters
  - non determinism choose
  - switch case term
Extend

- support for the extend of abstract domains

**extendExample**

abstract domain Product

- a domain can already contain some elements

static p1: Product

...

- and can be extended

- main rule r_main =

    extend Product with $x$ do skip
the interpreter supports seq rules

**SeqRule01**

```
main rule r_main =
  seq
  f := 10
  g := 5 + f
  f := 33
endseq
```

- at the beginning of every step
  - f is 33
  - g is 15!
Forall rule

ForRule01

signature:
controlled $f: \text{Prod}(\text{Integer, Integer}) \rightarrow \text{Integer}$

definitions:

main rule $r_{\text{main}} =$
forall $x$ in $\{1..3\}$ with $x \mod 2 = 1$ do
  forall $y$ in $\{5..9\}$ with $y \mod 2 = 0$ do
    $f(x, y) := x \times y$
Axioms

- the interpreter checks at every step if the axioms are satisfied or not
- In case they are not satisfied, it throws an Exception

**axioms**

axiom over `stockQuantity`:

\[(\forall p \in \text{Products with } stockQuantity(p) \geq 0)\]

- but the (faulty) machine does not check:

\[stockQuantity(p) := stockQuantity(p)\]
Inconsistent updates

- updates are checked if consistent
- **InconsistentUpdate**

par
choose $o$ in Orders with true do
  orderStatus($o$) := INVOICED
choose $oo$ in Orders with true do
  orderStatus($oo$) := CANCELLED
endpar
Flip Flop

- control state ASM

macro rule r_Fsm($i in State,$cond in Boolean, $rule in Rule,$j in State) =
  if ctl_state = $i and $cond then
    par
    $rule
    ctl_state := $j
  endif

main rule r_Main =
  par
  r_Fsm[ 0n, high, <<skip>>, 1n]
  r_Fsm[ 1n, low, <<skip>>, 0n]
endpar

circles as parameters
OrderSystem

- taken from

- it contains sequences, tuples, …

```java
// new orders to be inserted
monitored newOrders: Seq(Prod(Products, Natural))

// orders to be cancelled
monitored ordersToCancel: Seq(Orders)

// items to be inserted
monitored newItems: Seq(Prod(Products, Natural))
```
MultiAgent ASM

- the support for agents is still evolving
  - now the user has to set its scheduling policy in the main rule
- Example: philosophers
- version 0:
  - every philosophers can start eating \(\rightarrow\) inconsistent updates

```asm
macro rule r_Eat($self in Philosophers) =
  if (hungry($self)) then
    if ( isUndef(owner(left_fork($self))) and
      isUndef(owner(right_fork($self)))) then par
      owner(left_fork($self)) := $self
      owner(right_fork($self)) := $self
      eating($self) := true
    endpar
  endif
endif
```

version 1
- only one can eat

choose $p$ in Philosophers with hungry($p$) do
par
  r_Eat[$p$]
  r_Think[$p$]
endpar
MultiAgent ASM

- version 2
  - takes the fork only if it is free in two steps
  - if hungry he takes the left fork (if it is free)

```asm
macro rule r_Eat1($self in Philosophers) =
if (hungry($self) and
    isUndef(owner(left_fork($self)))) then
    owner(left_fork($self)) := $self
endif
```

- if he has the left fork, take the right (if it is free) and start eating

```asm
macro rule r_Eat2($self in Philosophers) =
if (owner(left_fork($self)) = $self and
    isUndef(owner(right_fork($self)))) then par
    owner(right_fork($self)) := $self
    eating($self) := true
endpar
```
version 3

- other scheduling policies can be defined:
- take two non adjacent philosophers

main rule r_all_philo =
choose $p1$ in Philosophers,
    $p2$ in Philosophers
with (($p1 != $p2) and (distance($p1,$p2) > 1))
do par
    r_program[$p1]
    r_program[$p2]
endpar
How to debug your ASM spec

- logger for tracing the parsing and the evaluation of terms and rules can be activated by changing the log4j properties file:

```java
log4j.logger.org.asmeta.interpreter.ReflectiveVisitor=WARN
log4j.logger.org.asmeta.interpreter.TermEvaluator=DEBUG
log4j.logger.org.asmeta.interpreter.RuleEvaluator=OFF
log4j.logger.org.asmeta.interpreter.TermSubstitution=INFO
log4j.logger.org.asmeta.interpreter.RuleSubstitution=OFF
log4j.logger.org.asmeta.interpreter=OFF
```
Integration of ATGT

- ATGT: abstract state machine generator tool
  - generates test cases by exploiting the counter example generation of Spin
  - reads asm gofer specs

- ATGT had its own internal representation of the ASMs

- how to make it AsmM complaint
  1. use the AsmM: change of everything
  2. write the XMI to asmgofer translator: is it really useful?
  3. add a pseudo parser which reads the AsmM model ad the asmgofer parser reads the file
ATGT made AsmM complaint

nothing has changed in the critical components

reads the model and build the right internal representation
Conclusions about the simulator

- confident that every non turbo single agent runs with the Asm Interpreter
  - some bugs still to be fixed
  - Agents need better treatment
  - 100+ examples
    - some for testing
    - many from the AsmBook

- very usable
  - graphical interface integrated in eclipse