

VMTS

Visual Modeling and Transformation System

T. Levendovszky, L. Lengyel, G. Mezei, H. Charaf, A Systematic Approach to Metamodeling Environments and Model Transformation Systems in VMTS, *Electronic Notes in Theoretical Computer Science* 127 (1) (2005) 65–75.

Dylan Kiss
University of Antwerp
dylan.kiss@student.uantwerpen.be

Metamodeling environment

- ▶ N-layer metamodeling environment
- ▶ Simplified UML class diagrams
- ▶ UML class diagram instantiation:
 - UML object diagram
 - UML class diagram
 - Metamodel of UML class diagram
- ▶ Two more layers:
 - Read-only meta-metamodel
 - Internal structure: labeled directed graph

Model storage

- ▶ AGSI
 - Attributed Graph Architecture Supporting Inheritance
- ▶ Every model can be a metamodel for others



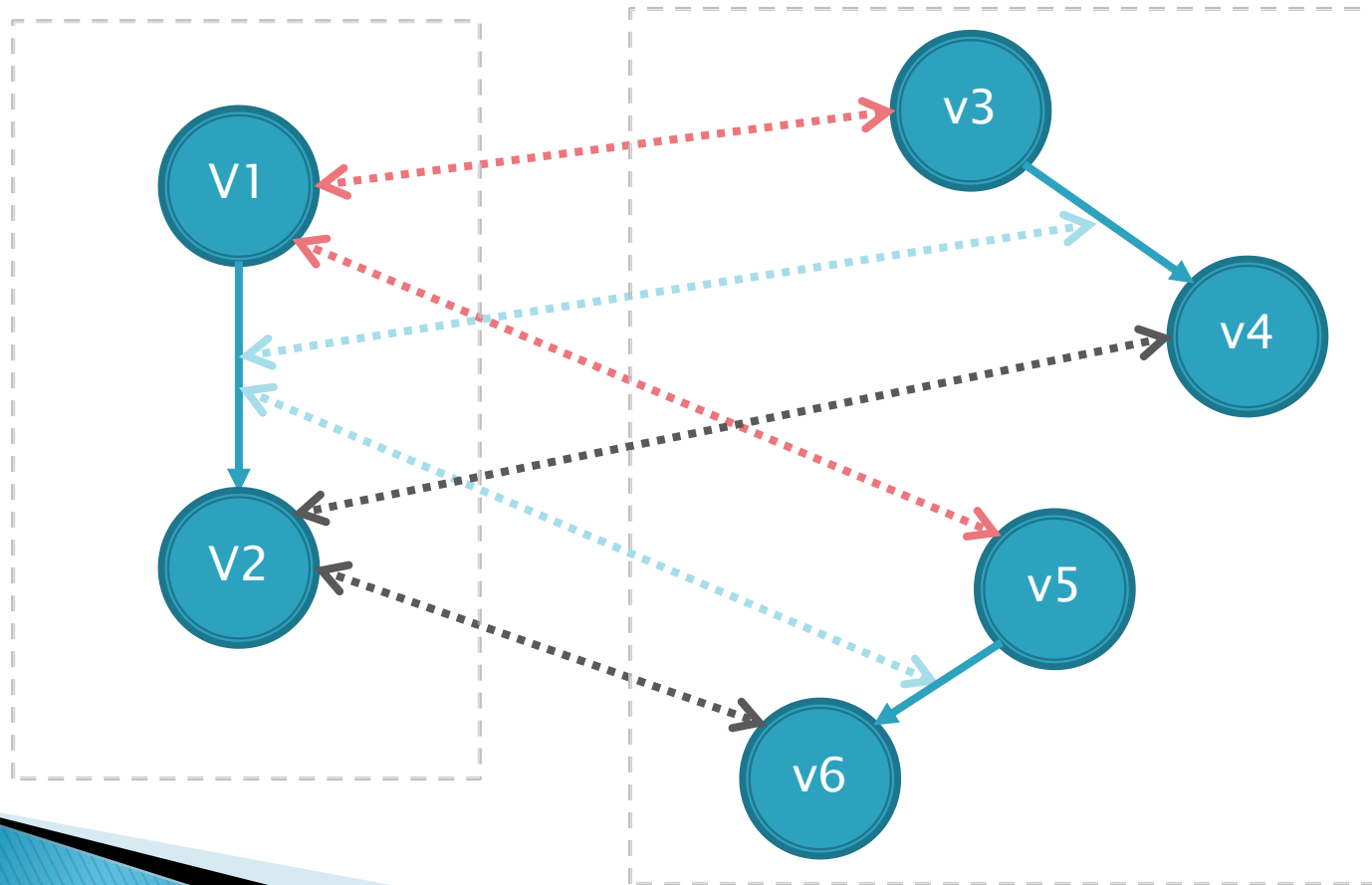
AGSI

- ▶ 3 basic graph constructs:
 - Nodes
 - Directed edges
 - Labels

- ▶ Metamodeling needs extra things:
 - Type-instance mapping
 - Containment
 - Inheritance
 - Association classes

AGSI

▶ Type-instance mapping



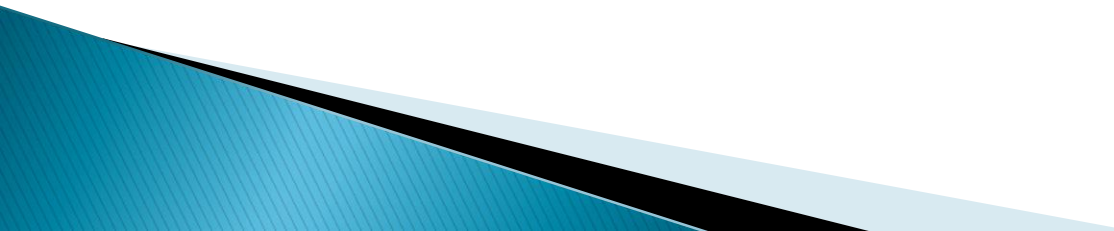
AGSI

- ▶ Containment
 - Parent–child bidirectional mapping
- ▶ Inheritance
 - Directed mapping from descendants to ancestors
- ▶ Association classes
 - Pseudo–nodes

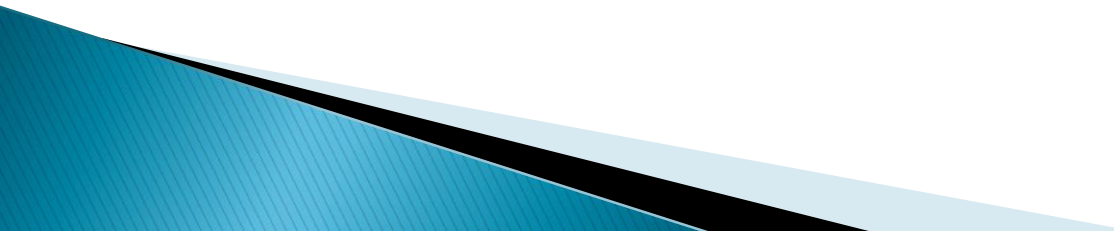
AGSI

- ▶ Model attributes (labels in directed graph)
 - Stored in XMI-like format
 - Meta-attributes that can be instantiated are stored in XSD file
 - Schema for XML file on instance level

Model transformations

- ▶ Traversing Model Processors
 - Create node
 - Connect nodes
 - Delete node
 - Delete edge
 - Set label
 - ▶ Regular objects in OO programming language
- 

Model transformation

- ▶ Visual Model Processors
 - Graph rewriting
 - Rules with LHS and RHS
 - ▶ Rules specified in terms of metamodel
 - ▶ Attribute transformation with XSLT scripts
- 

Modeling interface

The screenshot displays the VMTS Studio modeling interface. The main workspace shows a class diagram with two classes: **BookMeta** and **AuthorMeta**. **BookMeta** has an attribute `Title : string`, and **AuthorMeta** has an attribute `AuthorName : string`. A line connects the `Title` attribute of **BookMeta** to the `AuthorName` attribute of **AuthorMeta**, indicating a relationship. The interface includes a top menu bar with **Home**, **Palette**, **Constraints**, and **Model Transformation**. The **Palette** shows modes like **Drop Mode**, **Autoplace Mode**, and **Throw mode**, along with a **General** constraint category containing **Atom**, **Inheritance**, **Relationship**, and **Containment**. On the left, the **Project Explorer** shows a project structure with **BooksProject** and **BooksMetaBooksMeta**. On the right, the **Model Properties** window is open, showing a **Property Grid** with a **Filter** dropdown and a table with columns **Name** and **Value**. At the bottom, the **Error Window** shows **0 Errors**, **0 Warning**, and **0 Messages**, with buttons for **Check OCL** and **Check as Meta**. The **Output Window** has **Clear** and **Copy to Clipboard** buttons. Below the error and output windows are fields for **Related Model** and **Related Item**.

Modeling interface

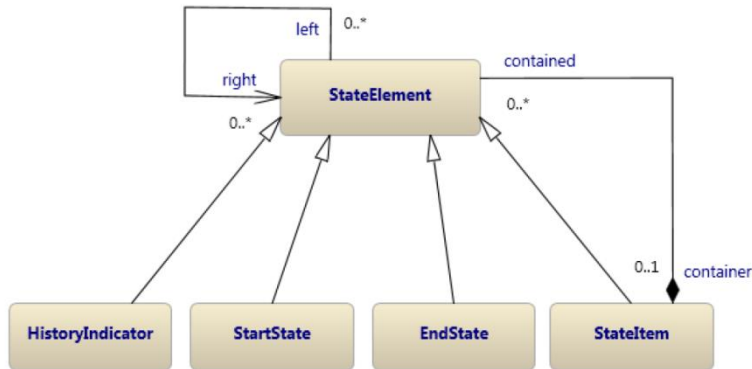
The screenshot displays the VMTS Studio modeling interface. The main workspace shows a class diagram with the following structure:

```
graph TD; Asimov --- Caliban; Asimov --- ShortStories; Lem --- ShortStories; Lem --- Solaris;
```

The interface includes several panels:

- Home / Palette / Constraints / Model Transformation:** Contains buttons for 'Drop Mode', 'Autoplace Mode', and 'Throw mode', along with a palette of elements like 'Book', 'Author', and 'BookAuthorRelationship'.
- Project Explorer:** Shows a tree view of the project structure, including 'BooksProject', 'Books', 'Diagrams', and various sub-diagrams like 'Asimov', 'ShortStories', 'Caliban', 'Solaris', and 'Lem'.
- Diagram on Books:** The central workspace displaying the class diagram.
- Model Properties:** A 'Property Grid' with a 'Filter' dropdown and a table with columns 'Name' and 'Value'.
- Error Window:** Shows '0 Errors', '0 Warning', and '0 Messages'. It includes buttons for 'Check OCL' and 'Check as Meta', and a table with columns 'Related Model' and 'Related Item'.
- Output Window:** Contains buttons for 'Clear' and 'Copy to Clipboard'.

Modeling interface



```
OCL C#
package Constraints
context StartState
inv start:
1 self.left->size() = 0
endpackage
```

Use OCL C#

OK Cancel

Model Properties	
Property Grid	
Filter	
Name	Value
[-]Attributes	
[AttributeMeta]	
[-][ComplexType]	
[-]ComplexType (Action)	
[-]Attribute	
[-]Attribute (Entry)	
IsReadOnly	False
Multiplicity	0..*
Name	Entry
TypeExpression	string Refresh
[-]Attribute (Exit)	
IsReadOnly	False
Multiplicity	0..*
Name	Exit
TypeExpression	string Refresh
[-]Attribute (Do)	
IsReadOnly	False
Multiplicity	0..*
Name	Do
TypeExpression	string Refresh
Name	Action

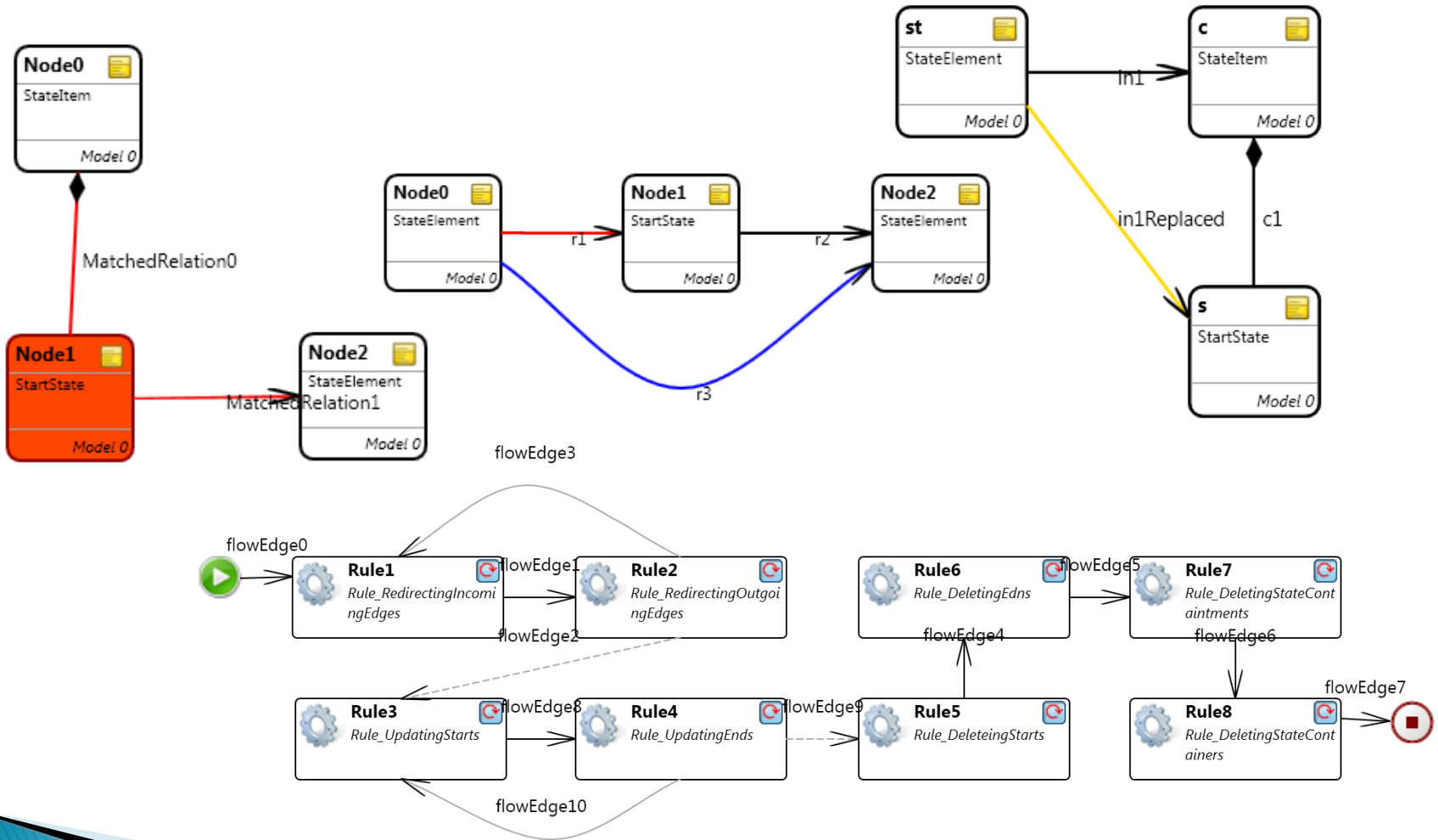
Modeling interface

The screenshot displays the VMTS Studio modeling interface. The main window shows a diagram titled "Diagram on LowyirDemo" at 100% zoom. The diagram is a state machine model with a root element "StateItem0" containing a "Containment" section. Inside "Containment", there are states "Normal", "Listening", and "Polite", and a "Start" state. Below "Containment" are "History" and "Kind" elements. "Normal" is connected to "Listening", "Listening" to "Polite", and "Start" to "Normal". "History" and "Kind" are connected to "Polite".

The interface includes a top menu bar with "Home", "Palette", "Constraints", and "Model Transformation". The "Palette" shows "Drop Mode", "Autoplace Mode", and "Throw mode" buttons, along with a palette icon and "All items" button. The "Constraints" and "Model Transformation" sections show "HistoryIndicator", "StartState", "EndState", "StateItem", "StateElement", and "Transition" elements.

On the right, the "Model Properties" panel is visible, showing a "Property Grid" with a "Filter" dropdown and a table with "Name" and "Value" columns. The bottom status bar shows "Output Window", "Error Window", "Project Explorer", and "Model Properties".

Transformation interface



Planned work

- ▶ Role-Playing Game modeling in VMTS
 - Metamodel (abstract syntax)
 - Concrete visual syntax
 - Transformation rules
 - Operational semantics
 - Denotational semantics
 - Compare with AToMPM and state advantages and disadvantages

Questions?

