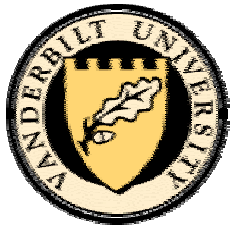
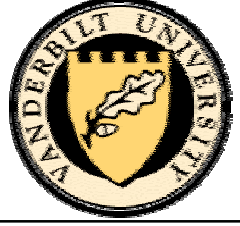


Institute for Software Integrated Systems
Vanderbilt University

A Meta-model based Model Transformation Tool: GReAT and Code Generator: the GReAT Compiler

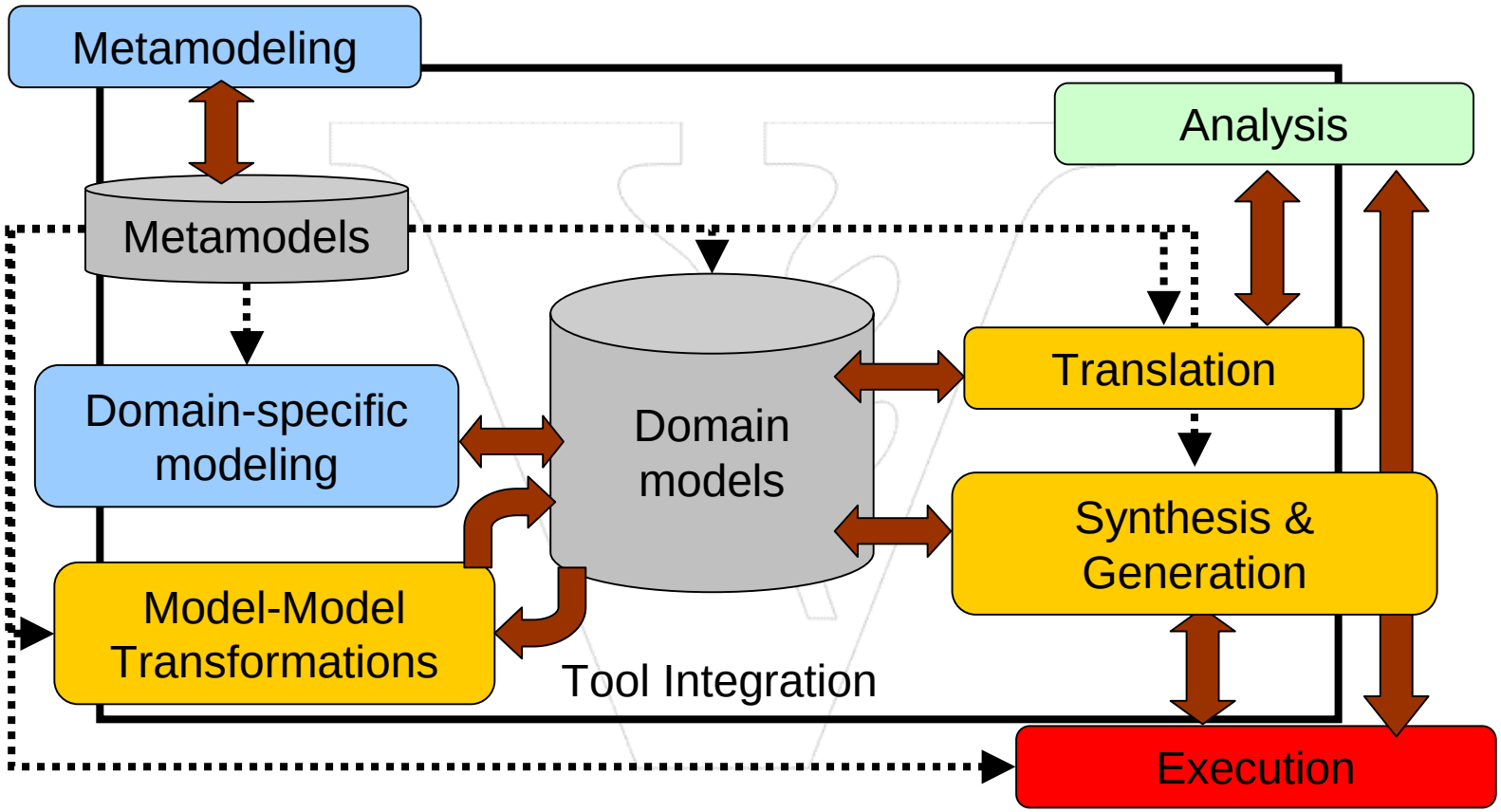
Attila Vizhanyo



Overview

- MIC Recap
- Model-to-model transformations as a GRT problem
- Universal Data Model (UDM)
- Graph Rewriting and Transformation (GReAT)
- Code Generator (CG)

MIC: Domain-Specific MDA

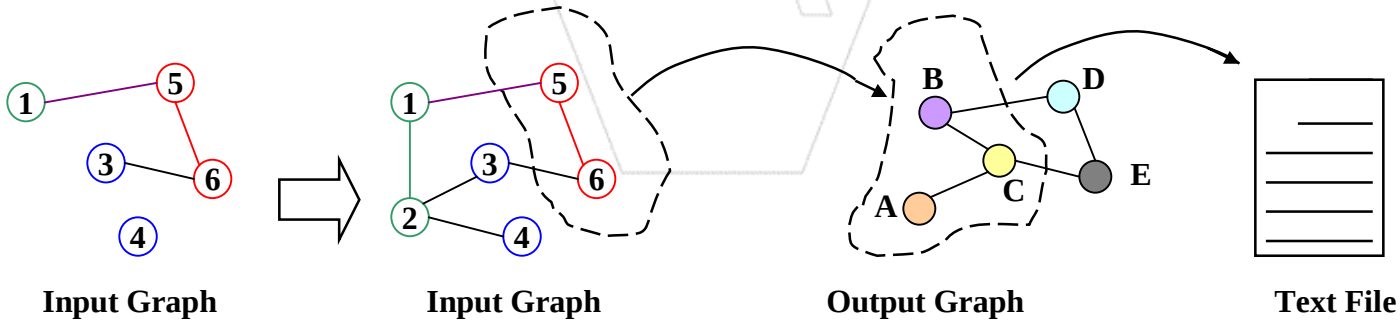


Design challenges of the generic model transformation tool

- Precise specification on categories of models
  - Categories= Domains (= Metamodels in MIC)
- The input & output models typically conform to different metamodels
  - Support heterogeneous domains
  - Cross references between domains (temporary domain)
- Automated generation of executable model transformer code from the specification
- Acceptable runtime performance
- Simple semantics, usability

Graph Transformations

- Represent models with graphs
- Graph grammars and transformations
  - Over 30 years of research -- Node & Hyper edge replacement -- Algebraic approaches -- Programmed graph replacement systems. Systems: PROGRES, AGG
- Graph transformations can be used to generate domain-specific model transformers
- Traditional transformation approaches cannot be applied directly



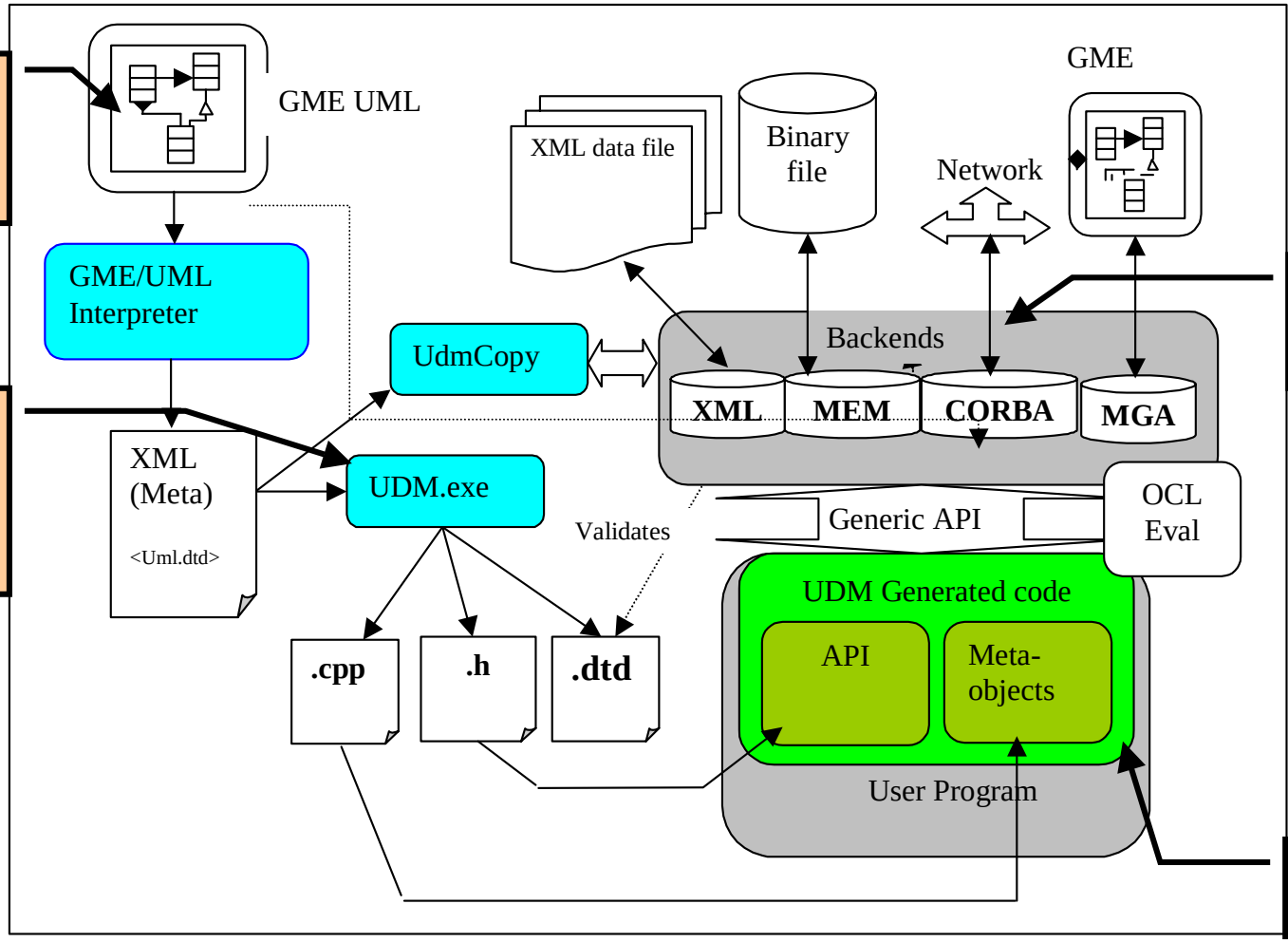
The UDM Framework

Programmatic manipulation of graphs

Analyze graph domains for type safety

Metamodeling of data: UML class diagrams

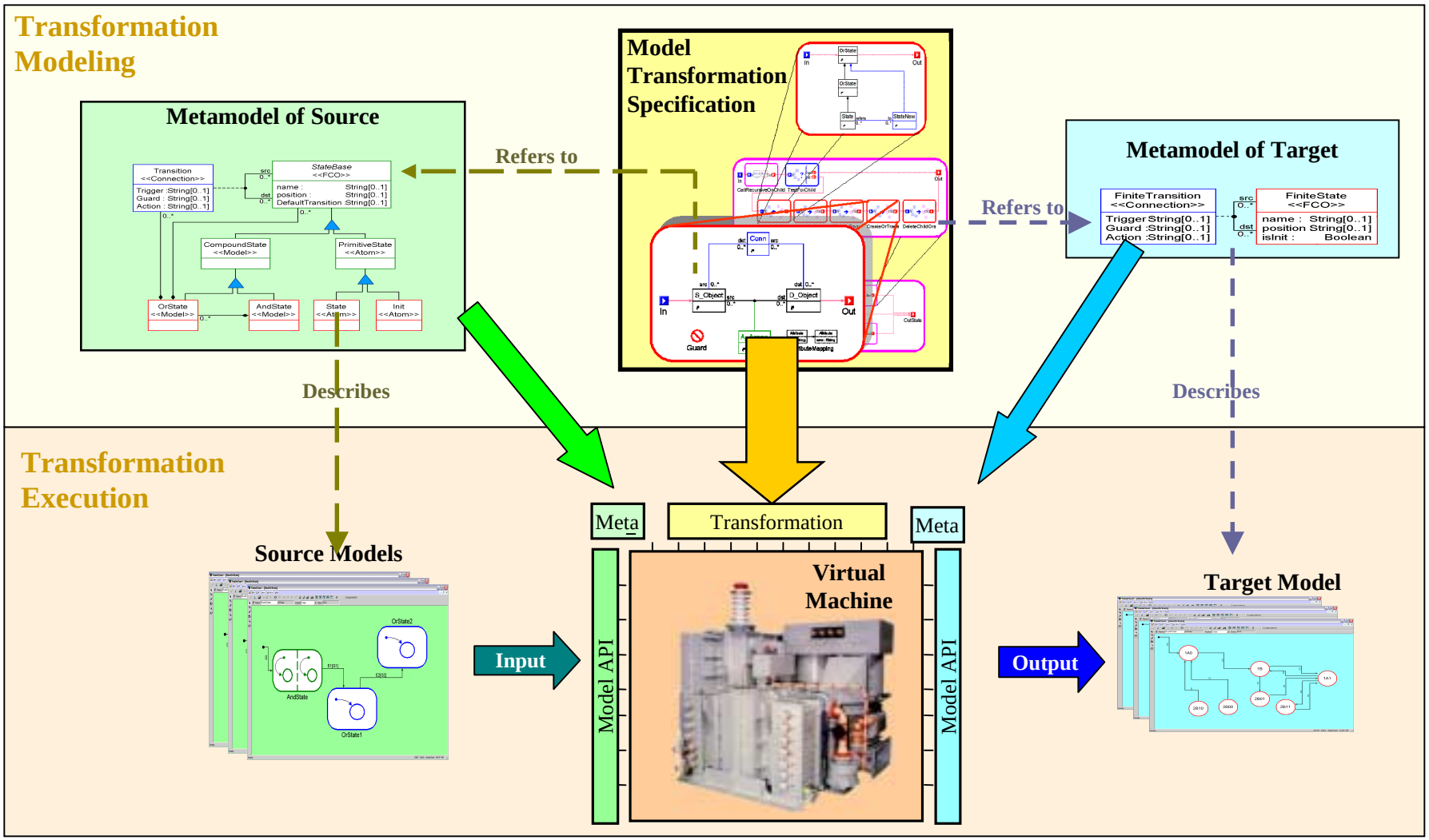
API Code Generator: Builds C++ "handle" classes



Backends: Generic object implementation

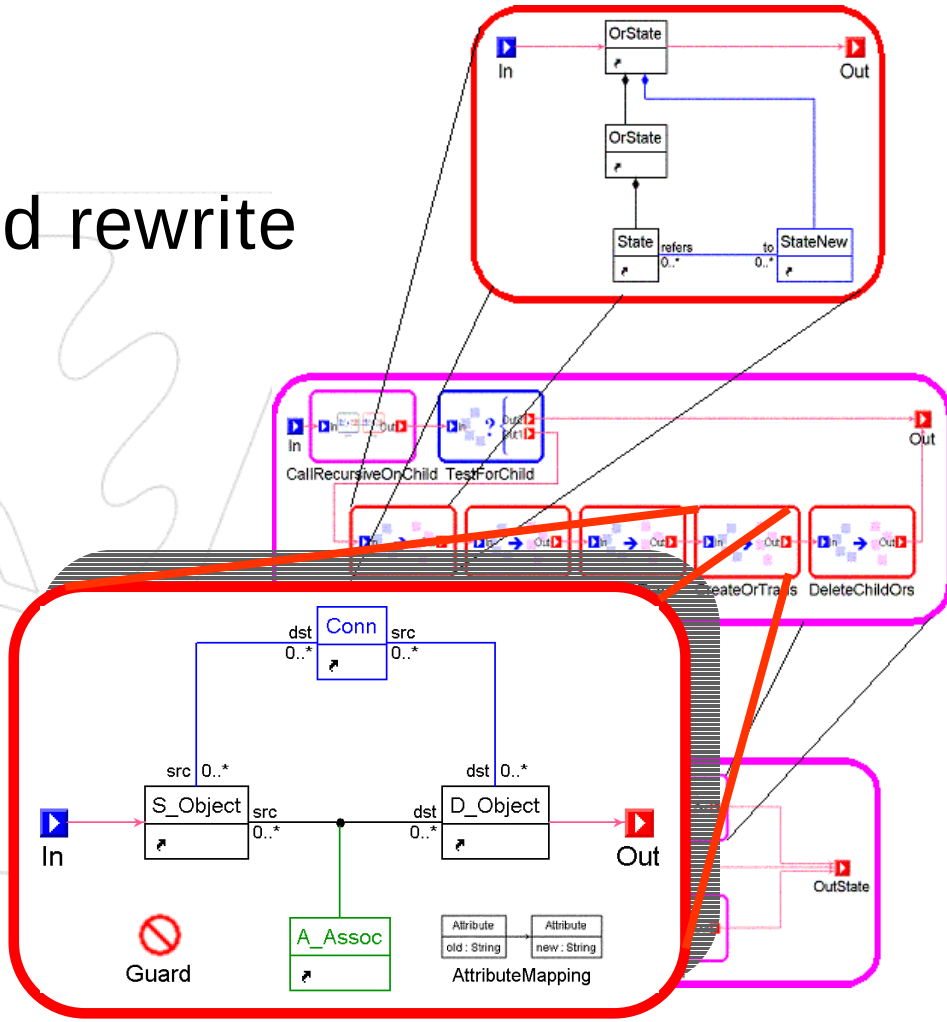
Reflection: Meta-objects

Graph Rewriting & Transformation (GReAT) Overview

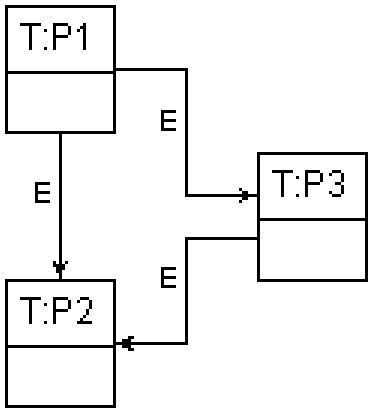


The GReAT Language

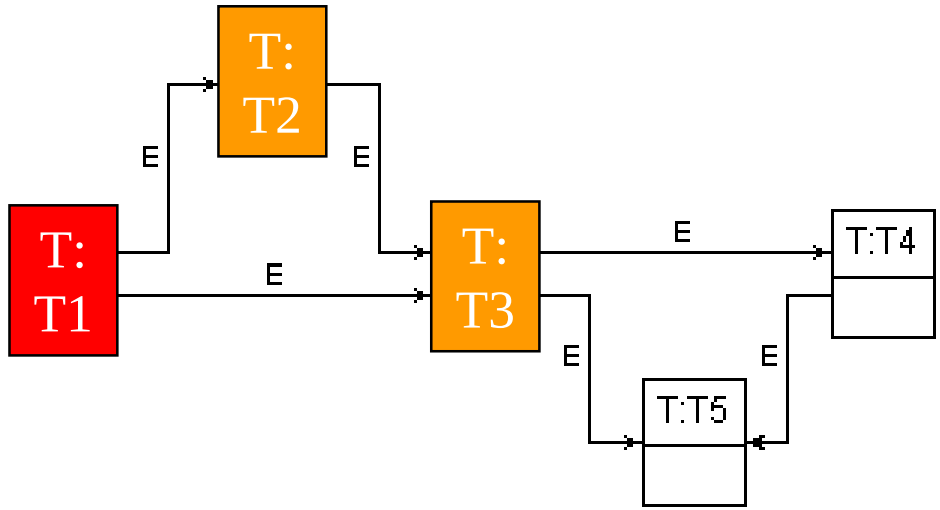
- Pattern specification
  - Patterns with cardinality
- Graph transformation and rewrite
  - Create New Objects
  - Delete Objects
  - Modify Attributes
- High-level control flow
  - Hierarchy
  - Sequencing
  - Recursion
  - Branching



Pattern specification language & Pattern matching



Pattern graph

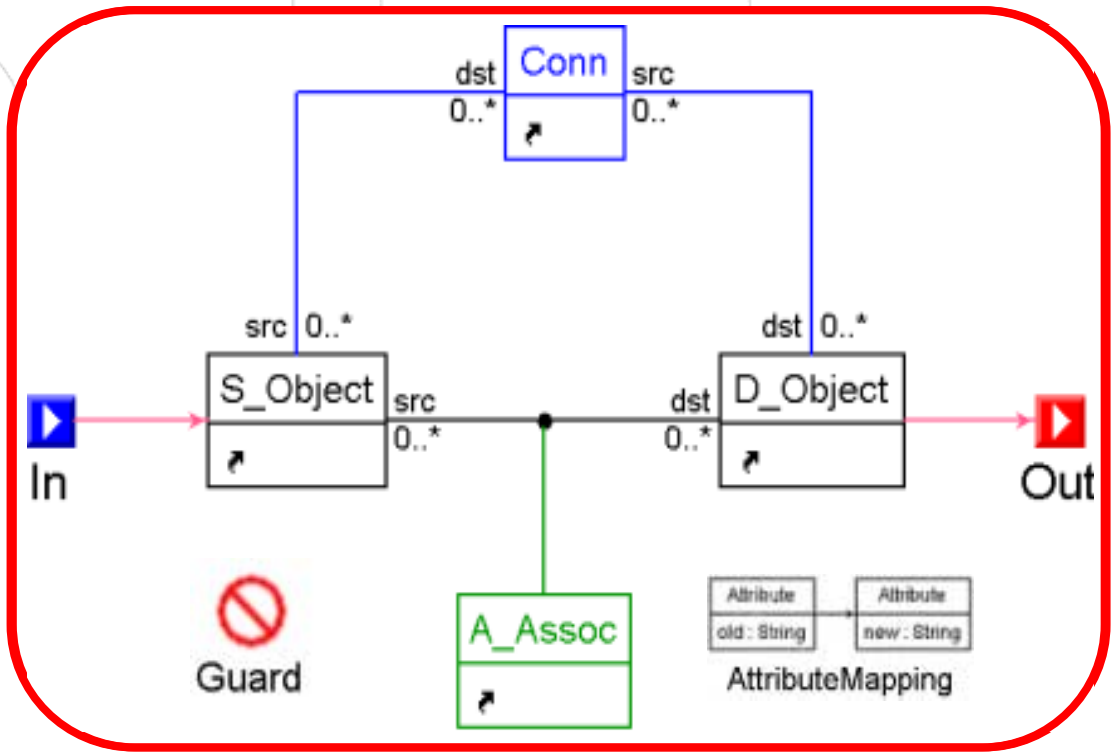


Host graph

- Time complexity is exponential in the number of pattern vertices and edges
- Optimization: start with at least one known vertex "pivoted pattern matching"

Graph transformation language

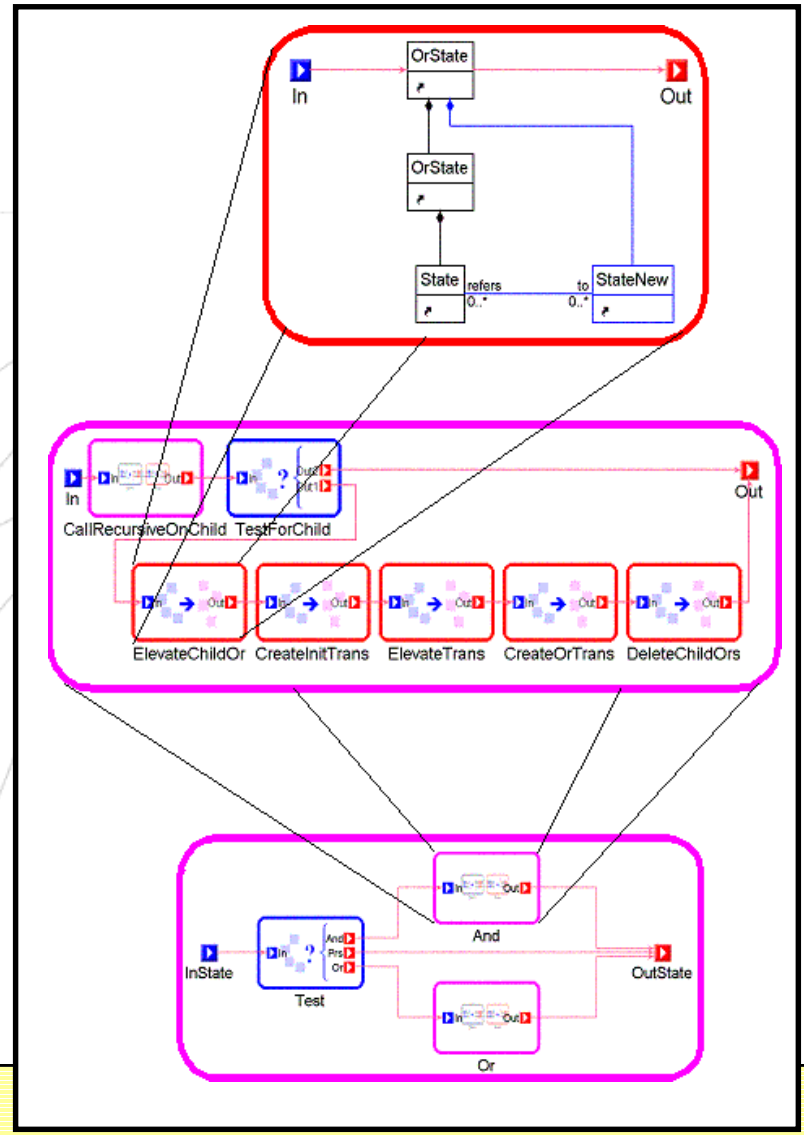
- Production (Rule)
  - Basic transformation entity
  - UML notation
  - Ports: “Reusing previously matched objects”



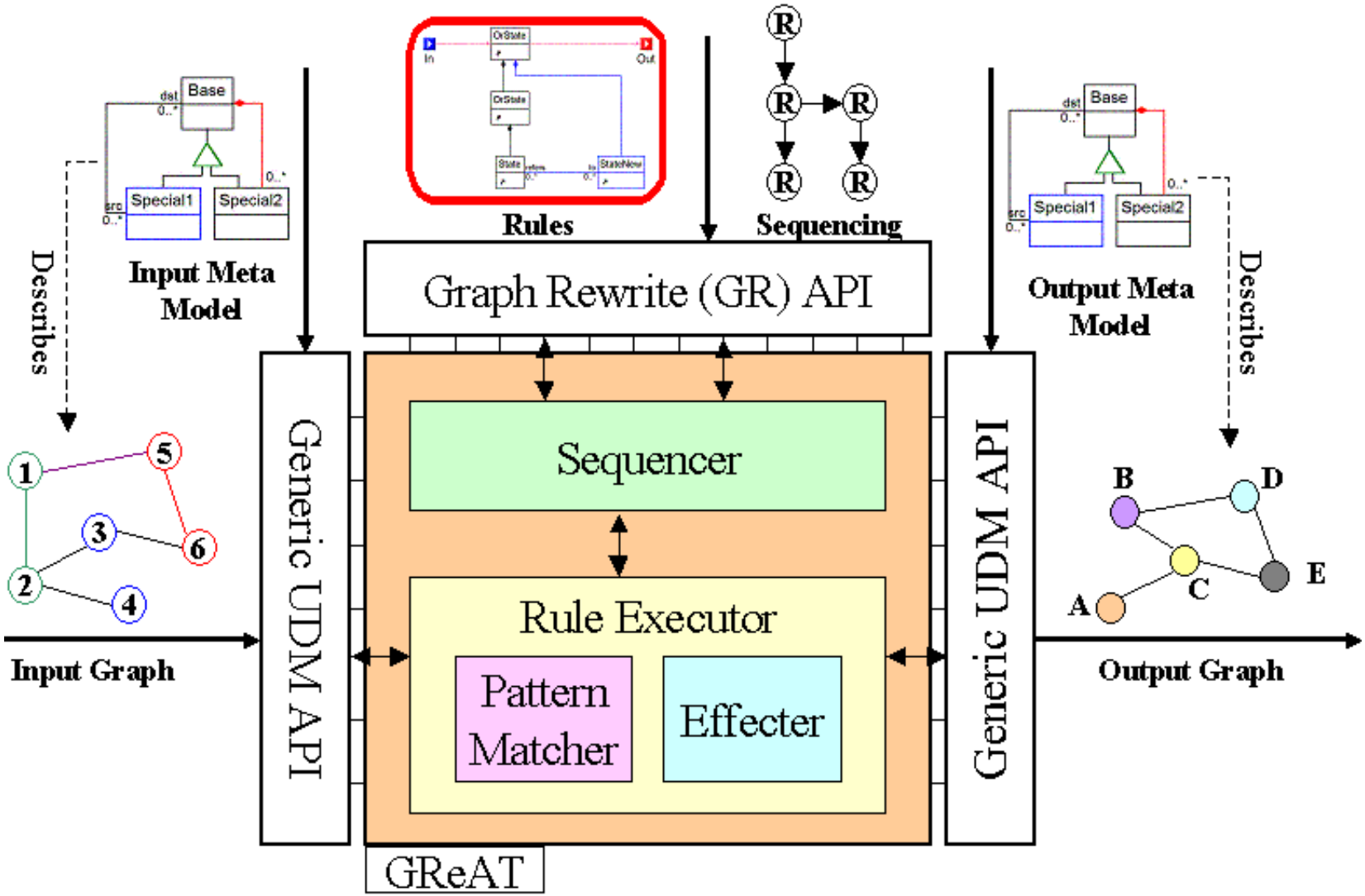
- Find
- Find and delete
- Create New

Control Flow Language

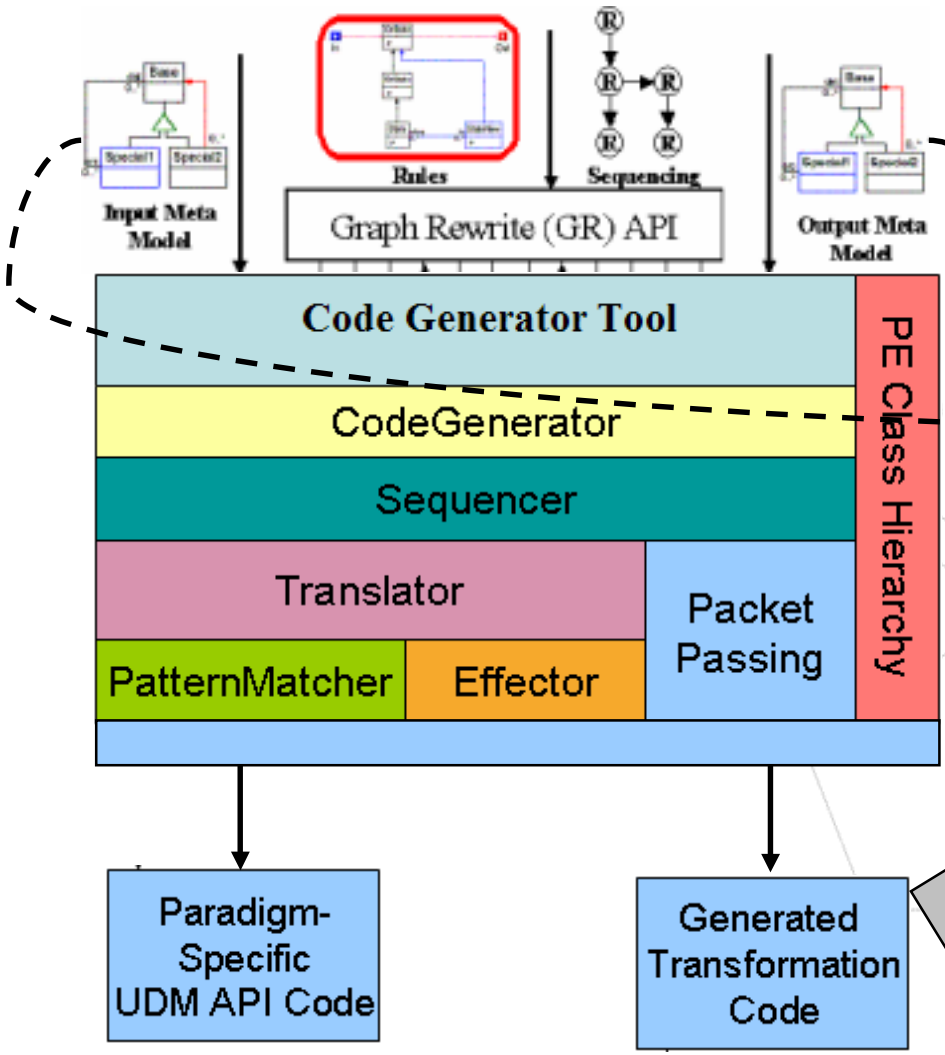
- Sequencing of rules necessary for efficiency
  - Explicit control flow
  - Simple semantics
  - Allows reasoning about transformation
- Control Constructs
  - Sequencing
  - Branching
  - Hierarchical nesting
  - Recursion



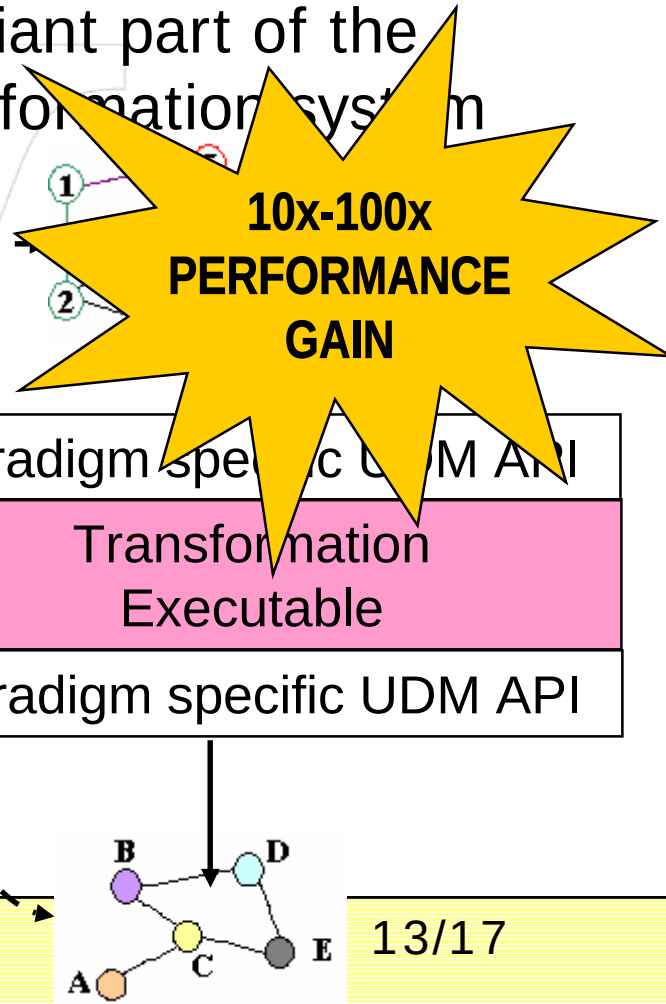
GReAT Tools: Transformation Engine



GReAT Tools: Code Generator (1/2)

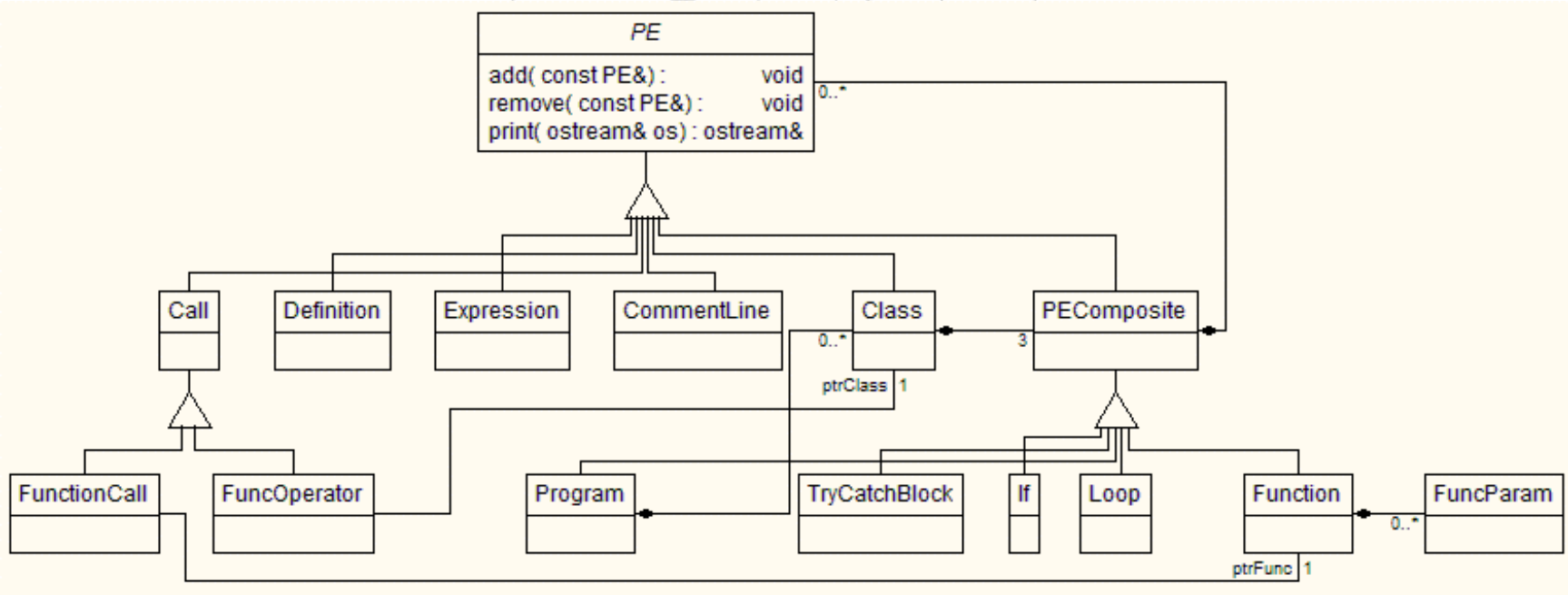


- Transformation and the metamodels make up the invariant part of the transformation system



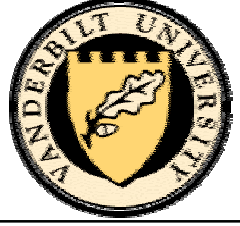
GReAT Tools: Code Generator (2/2)

- Each rule is compiled into a C++ class
- Ports are implemented with function argument list
- Efficient pattern matching with “identity checks”
- Quasi C++ syntax tree used to build the code



GReAT in Action

Problem	Developer	GReAT		Hand code
		Primitive Rules # / Compound Rules #	Man hours	LOC
Hierarchical Data Flow (HDF) to Flat Data Flow (FDF)	Feng Shi	11/3	~3	~200
KHORUS to GUDML	Abdullah Sowayan	19/10	~8	~500
Hierarchical Concurrent State Machine (HCSM) to Finite State Machine (FSM)	Aditya Agrawal	21/5	~8	~500
Simulink Stateflow to C code	Sandeep Neema	70/50	~25	~2.5K
Matlab Simulink/ Stateflow to Hybrid Automata	Aditya Agrawal	154/43	~60	~5K



Conclusion

- In Domain Specific MDA there are many models & implementations:

  Model Transformations are essential!

- GReAT for model transformations
  - A metamodel based transformation language using graph rewriting and transformations that support multiple graphs (that may belong to different domains) with an efficient implementation is suitable for the specification of model transformers.
- CG generates efficient code
- Increase productivity
  - Decrease the development time of model transformers

Closing Remarks

Key Publications

Agrawal A., Karsai G., Ledeczi A.: **An End-to-End Domain-Driven Software Development Framework**, 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), Domain-Driven Development Track, Anaheim, CA, October, 2003.

Karsai G., Agrawal A., Shi F., Sprinkle J.: **On the Use of Graph Transformations for the Formal Specification of Model Interpreters**, Journal of Universal Computer Science, Special issue on Formal Specification of CBS, 2003.

Agrawal A., Karsai G., Shi F.: **Graph Transformations on Domain-Specific Models**, Technical report, ISIS, Vanderbilt University, Nashville, TN, 2003.

Vizhanyo A., Agrawal A., Shi F.: **Towards Generation of High-performance Transformations,** Generative Programming and Component Engineering, Vancouver, Canada, October 24, 2004.

GReAT and UDM Download URL:
http://www.isis.vanderbilt.edu/projects/mobies/downloads.asp