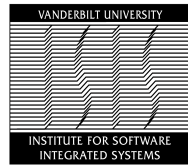


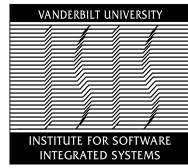
Towards Formalizing Domain-specific Modeling Languages

Kai Chen

Janos Sztipanovits

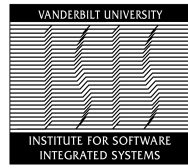
Sandeep Neema

Outline



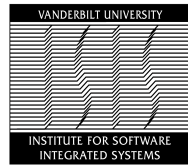
- DSML overview
- Framework for DSML design
  - Syntax definition
  - Semantic domain specification
  - Semantic mapping specification
- MoC & semantic anchoring
- Case study
- Conclusion
- Future Plans

Domain-specific Modeling Language



- What is a domain?
  - For MIC, a domain is defined by interaction patterns among physical and computational components of a family of computer-based systems.
- What is a DSML?
  - A DSML is a modeling language that is tailored to particular constraints and assumptions of an application domain .
  - The domain concepts are represented by language primitives, so the system can be modeled as it exists.

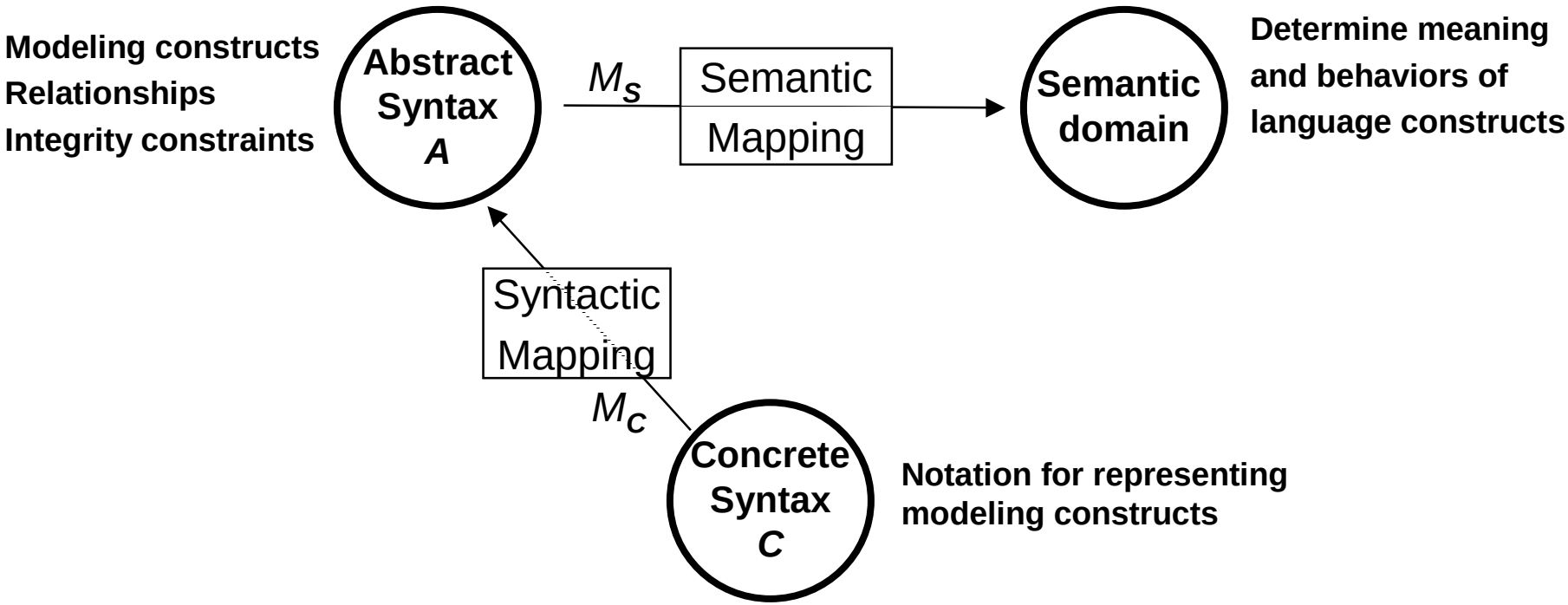
Current Challenges for DSML



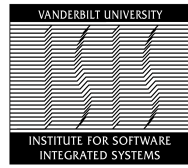
- These is no formalized design process for the DSML.
  - Comparing with other popular programming languages, a DSML experiences less research and tests. The ad-hoc design process for the DSML makes it even more dangerous.
- The semantics definition for the DSML is weak.
  - Currently, the semantics of a DSML is often implicitly defined by its model interpreter, and additional documents written in a natural language are used to explain it.
- The “domain-specific” limits the reusability of the DSML.
  - A small change in the domain features may destroy the chance to reuse a whole predefined DSML.

Formal DSML Definition

$$DSML = < C, A, M_C, S, M_S >$$

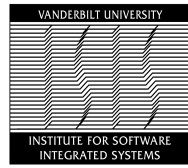


Outline



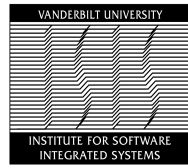
- DSML overview
- Framework for DSML design
  - Syntax definition
  - Semantic domain specification
  - Semantic mapping specification
- MoC & semantic anchoring
- Case study
- Conclusion
- Future Plans

Syntax Definition



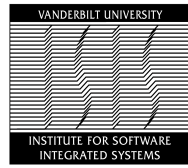
- Abstract syntax
  - Modeling concepts, their relationships, and integrity constraints.
- Concrete syntax
  - Notations (graphical, textual or mixed) to express elements in the abstract syntax.
- Syntactic mapping
  - Relate syntactic notations to elements in the concrete syntax.

Metamodel



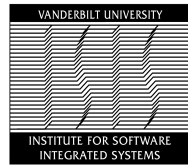
- Metamodel can provide all information needed to define the syntax of a DSML.
- MOF or UML class diagrams
  - Define modeling concepts and their relationships.
- OCL constraints
  - Define well-formedness rules.
- Visualization specification
  - Specify how domain models are to be visualized in a visual modeling environment.

Outline



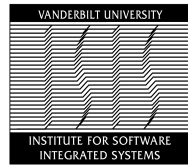
- DSML overview
- Framework for DSML design
  - Syntax definition
  - Semantic domain specification
  - Semantic mapping specification
- MoC & semantic anchoring
- Case study
- Conclusion
- Future Plans

Concepts



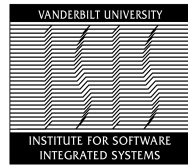
- ***Semantics of a domain model***
  - The meaning or behaviors of a domain model.
- ***Semantics of a DSML***
  - Determines meaning or behaviors of all its legal modeling constructs. Includes a semantic domain and a semantic mapping.
- ***Semantic domain***
  - A domain that provides a set of expressions that have well-defined meaning or behaviors.
- ***Semantic mapping***
  - Relates the abstract syntax to the elements in the semantic domain.

Structural vs. Behavioral Semantics



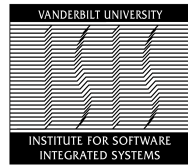
- ***Structural semantics***
  - Describe mathematical structures of a domain model.
- ***Behavioral semantics***
  - Determines behaviors of a domain model.

Semantic Domain Specification



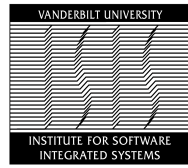
- To specify the semantic domain of a DSML, there should be an existing semantic domain that provides expression with well-defined semantics.
- This existing semantic domain is called the ***fundamental semantic domain*** (FSD) for the DSML.

Requirements for FSD candidates



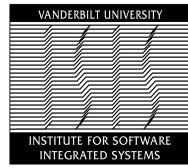
- Be mathematically precise.
- Be sufficient rich and flexible enough to cover a wide variety of application domains.
- Be scalable to support appropriate abstraction mechanisms.
- Have good intelligibility to avoid formalization overhead as far as possible.
- It is better for the semantic domain to be executable with readily supporting tools.

Why ASM?



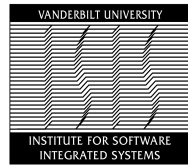
- Precision
  - ASM have a solid mathematical foundations.
- Generality
  - Successful application in a wide variety of domains: sequential, parallel, and distributed systems; abstract-time and real-time systems; finite-state and infinite-state domains.
- Scalability
  - Describe a system at several different layers of abstraction.

Why ASM? (cont'd)



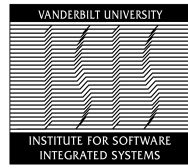
- Understandability
  - Easy to read and write.
- Executability
  - A specification methodology which is executable allows one to test for errors in the specification.
- Success in semantics definition
  - Successful experiences in the semantics definition for languages, such as C, C++, COBOL, Java, Oberon, Prolog, SDL and VHDL.

Outline



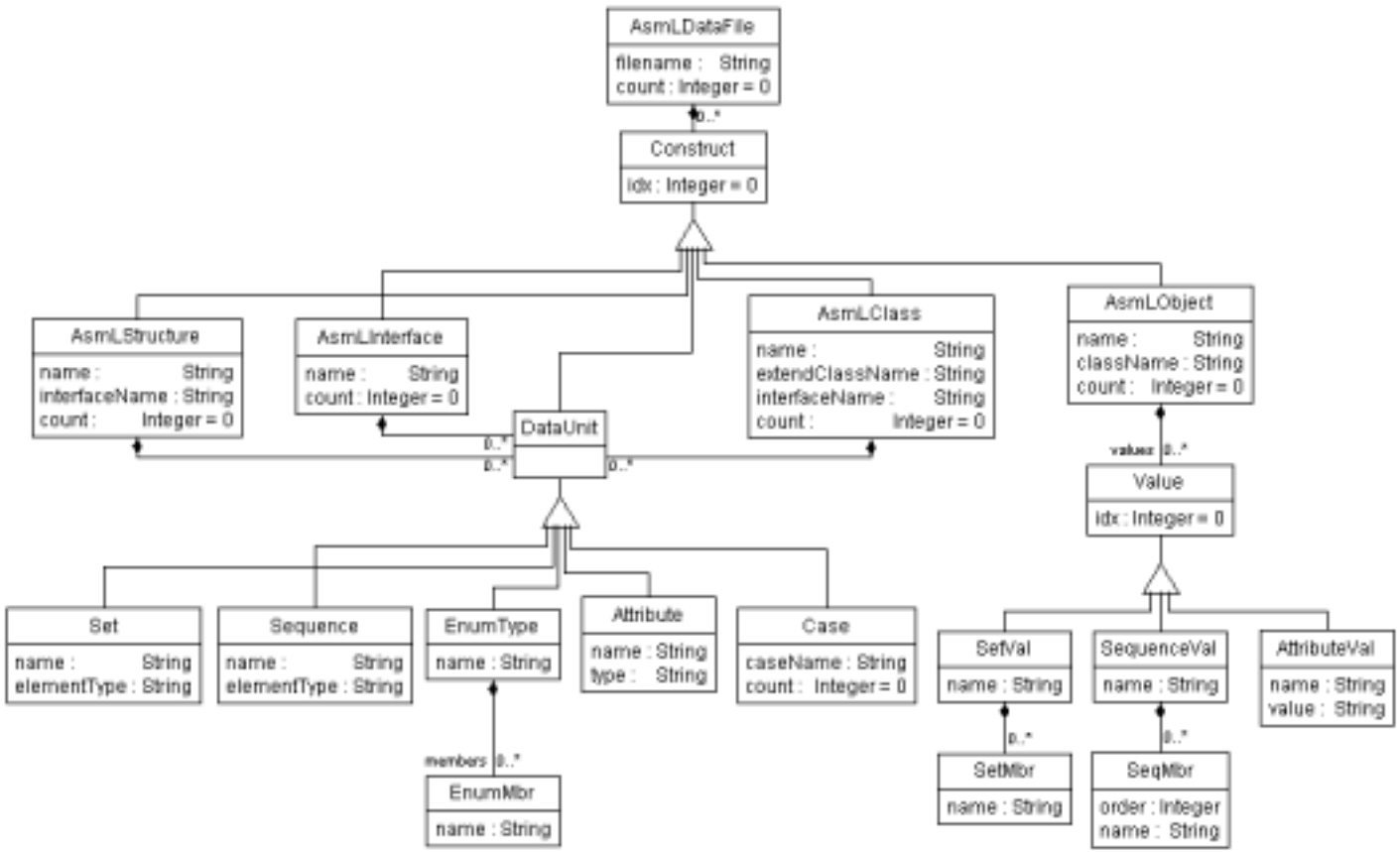
- DSML overview
- Framework for DSML design
  - Syntax definition
  - Semantic domain specification
  - Semantic mapping specification
- MoC & semantic anchoring
- Case study
- Conclusion
- Future Plans

Semantic Mapping



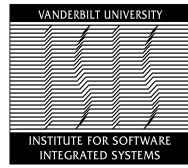
- The syntactic notations provided by the FSD are independent of the syntax defined by the DSML itself.
- A semantic mapping, $M_S: A \rightarrow S$, is a bridge connecting the syntax of a DSML with its semantic domain.

Metamodel for a Set of AsmL Abstract Data Structures



- AsmL specifications include two parts: data structures and execution algorithm specifications.
- It is much easier to model only the AsmL data structures instead of the whole syntax of the AsmL.

Tool : GReAT

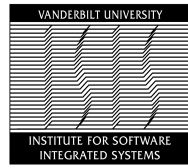


- The semantic mapping specification consists of a sequence of mapping rules specified in the GReAT language.
- Source metamodel
  - Metamodel for a DSML.
- Destination metamodel
  - Metamodel for a set of AsmL abstract data structure.

DSML Design Framework

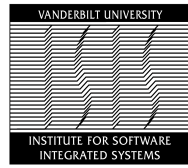
- Step 1
  - Define the DSML metamodel <*A, C, M_c*> (GME).
- Step 2
  - Specify the DSML semantic domain *S* using AsmL (AsmL tools).
- Step 3
  - Specify the DSML semantic mapping M_s, relating the metamodel with elements in the semantic domain using the GReAT language (GReAT).

Outline



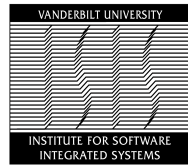
- DSML overview
- Framework for DSML design
  - Syntax definition
  - Semantic domain specification
  - Semantic mapping specification
- MoC & semantic anchoring
- Case study
- Conclusion
- Future Plans

Models of Computation



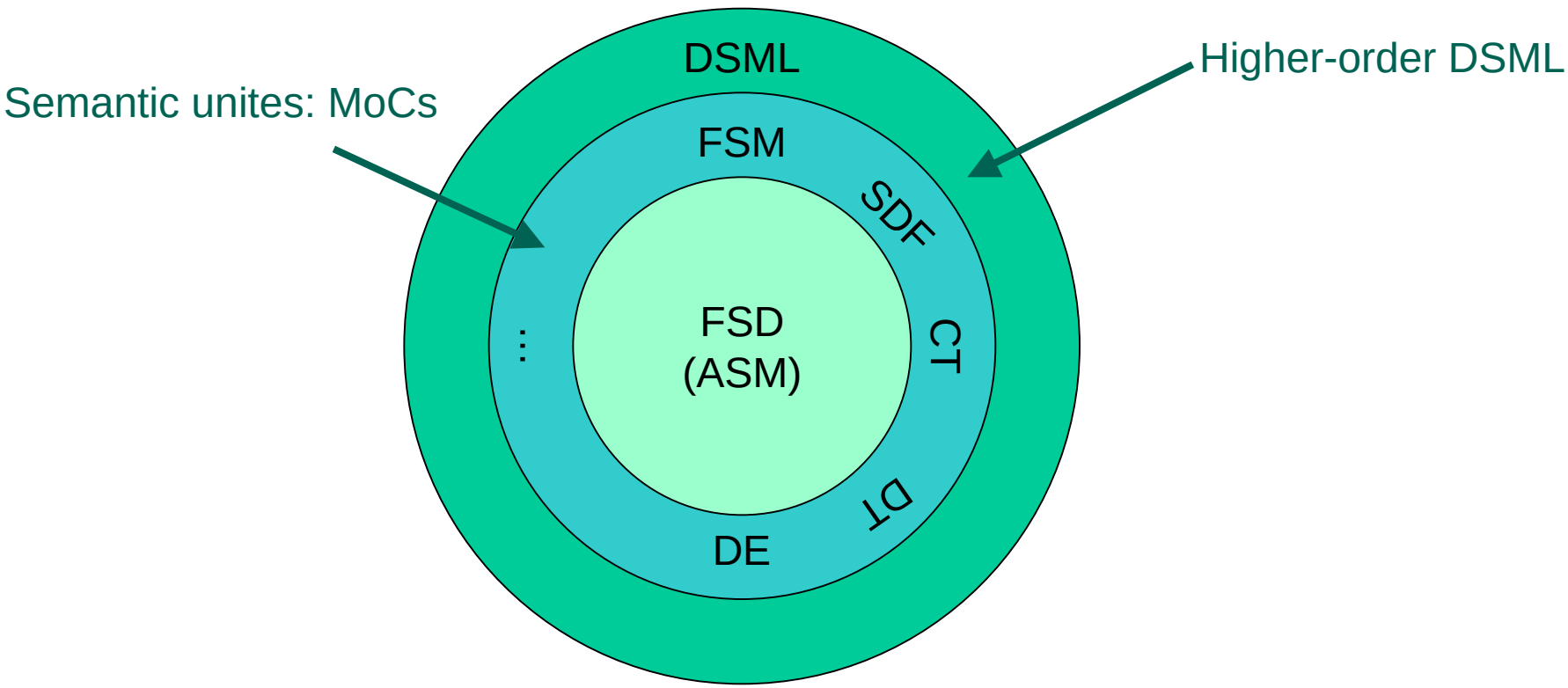
- In the embedded software industry, there is a set of standard, well-known semantic domains, which describe canonical interaction patterns among physical and computational components.
- ***Models of computation*** (MoCs) is a set of well-defined DSMLs that capture these standard, well-known semantic domains.

Semantic Anchoring



- Intention
  - Simplify the DSML design process
- MoCs can be used as reusable ***semantic units***, with which the semantics of higher-order DSMLs are able to be specified through ***semantic anchoring*** to these semantic units.

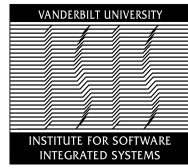
Semantic Domain Architecture



DSML Design Framework with Semantic Anchoring

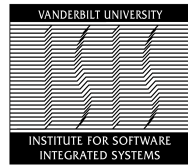
- Step 1
  - Define the DSML metamodel <*A, C, M_C*> (GME).
- Step 2
  - Select a proper MoC $L_i =< A_i, C_i, M_{Ci}, S_i, M_{Si} >$ as a semantic unit (MoC library).
- Step 3
  - Specify the semantic anchoring $M_A = A \rightarrow A_i$ (GReAT).
  - Now, DSML $L =< A, C, M_C, S_i, M_A \square M_{Si} >$.

Outline



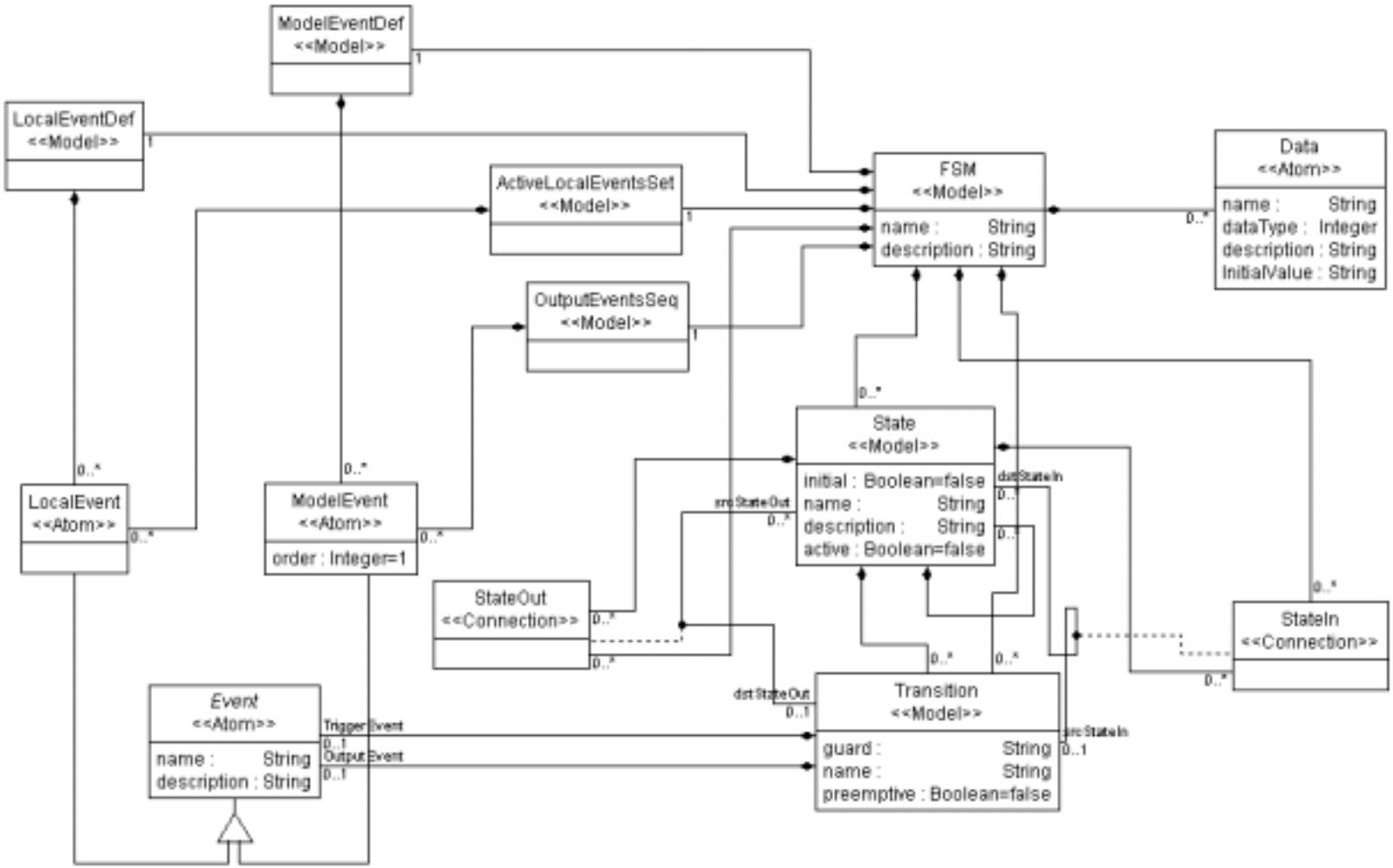
- DSML overview
- Framework for DSML design
  - Syntax definition
  - Semantic domain specification
  - Semantic mapping specification
- MoC & semantic anchoring
- Case study
- Conclusion
- Future Plans

Finite-state Machine Domain in Ptolemy II

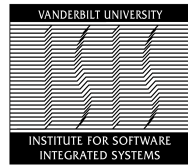


- A Statecharts variant.
- Define a DSML called finite-state machine modeling language (FML) to capture the FSM domain in Ptolemy II.

Metamodel for FML



Semantic Domain Specification



- Represent FML in AsmL data structure.
- Specify execution algorithm in a step-for-step manner.

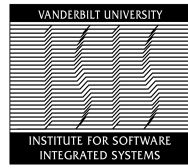
AsmL data Structure

- Event & FSM class

```
interface Event
structure ModelEvent implements Event
structure LocalEvent implements Event

class FSM
  var outputEvents as Seq of ModelEvent
  var localEvents as Set of LocalEvent
  var initialState as State
  var children as Set of State
```

AsmL data Structure (cont'd)

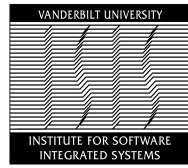


- State & Transition class

```
class State
  var active as Boolean = false
  var initial as Boolean
  var initialState as State?
  var parentState as State?
  var slaves as Set of State
  var outTransitions as Set of Transition

class Transition
  var guard as Boolean
  var preemptive as Boolean
  var triggerEvent as Event?
  var outputEvent as Event?
  var src as State
  var dst as State
```

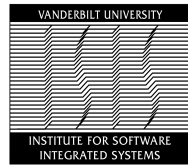
Execution algorithm



- Top-level FSM model reaction

```
fsmReact (fsm as FSM, e as ModelEvent) =
  step
    let cs as State = getCurrentState (fsm, e)
  step
    let pt as Transition? = getPreemptiveTrasition (fsm, cs, e)
  step
    if pt <> null then
         doTransition (fsm, cs, pt)
    else
      step
        if isHierarchicalState (cs) then invocateSlaves (fsm, cs, e)
      step
        let npt as Transition? = getNonpreemptiveTranstion (fsm, cs, e)
      step
        if npt <> null then doTransition (fsm, cs, npt)
```

Execution algorithm (cont'd)



- Do transition

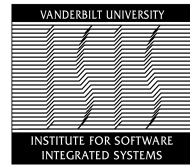
```
doTransition (fsm as FSM, s as State, t as Transition) =
  require s.active
  step exitState (s)
  step if t.outputEvent <> null then emitEvent (fsm, t.outputEvent)
  step activateState (fsm, t.dst)
```

Execution algorithm (cont'd)

- Activate state

```
activateState (fsm as FSM, s as State) =
  step s.active := true
  step
    if isAtomicState (s) then
      let t as Transition? = getInstantaneousTransition (s)
      step if t <> null then doTransition (fsm, s, t)
```

Execution algorithm (cont'd)

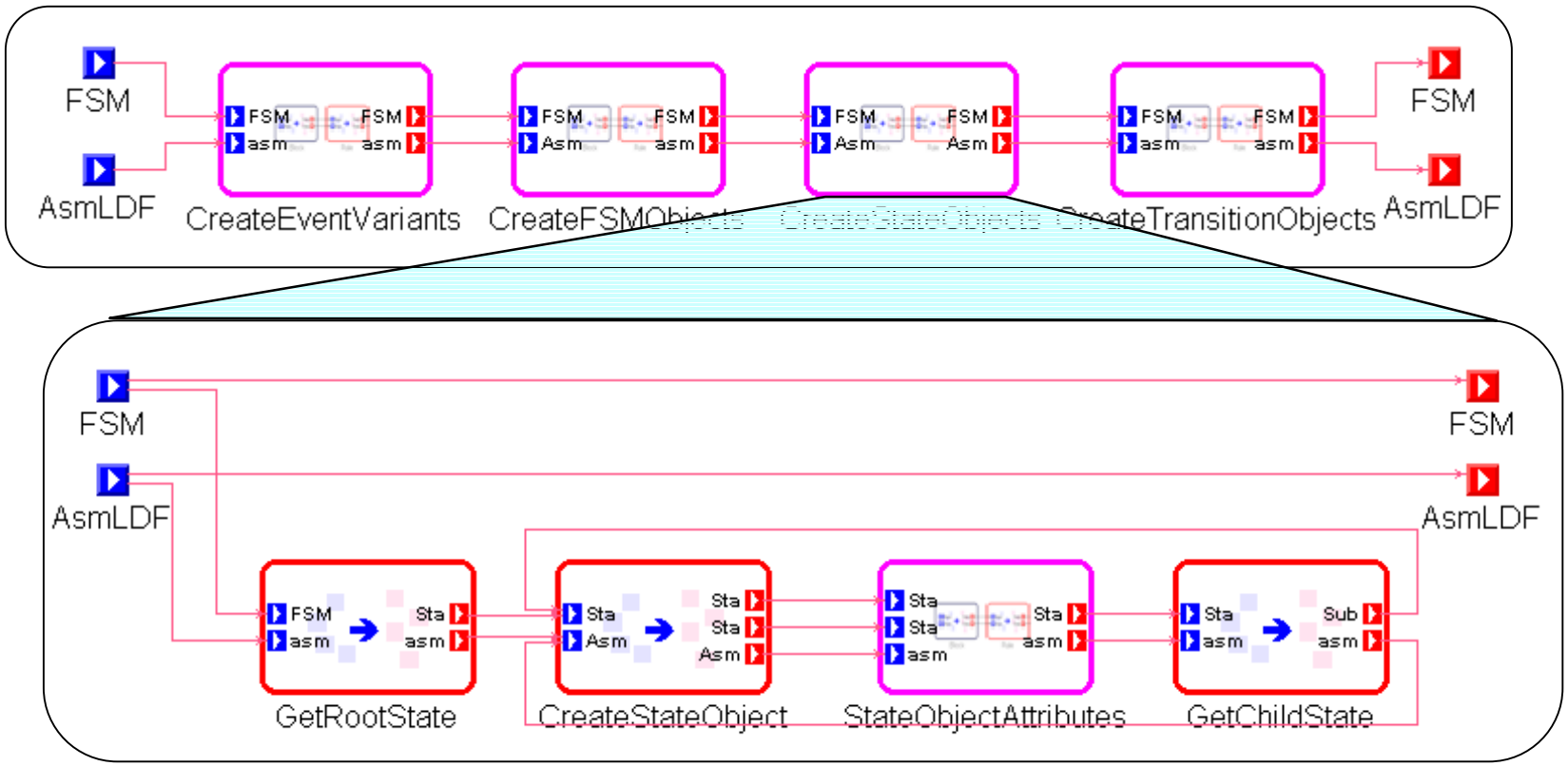


- Get instantaneous transition

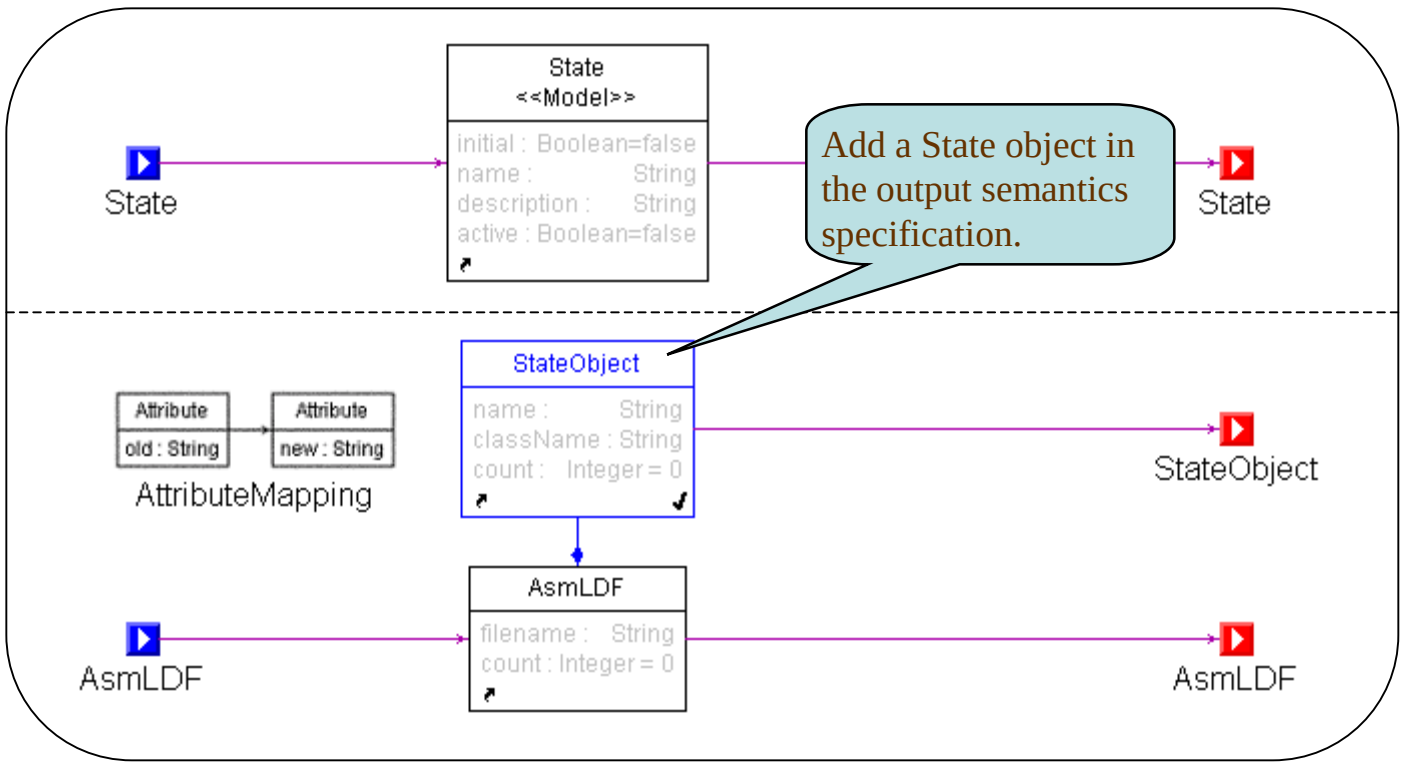
```
getInstantaneousTransition (s as State) as Transition? =
  require isAtomicState (s)
  step
    let ts = {t|t in s.outTransitions where t.triggerEvent = null
    and t.guard }
  step if Size (ts) > 1 then error "non-deterministic error"
  step
    choose t in ts
      return t
    ifnone
      return null
```

Semantic Mapping Specification

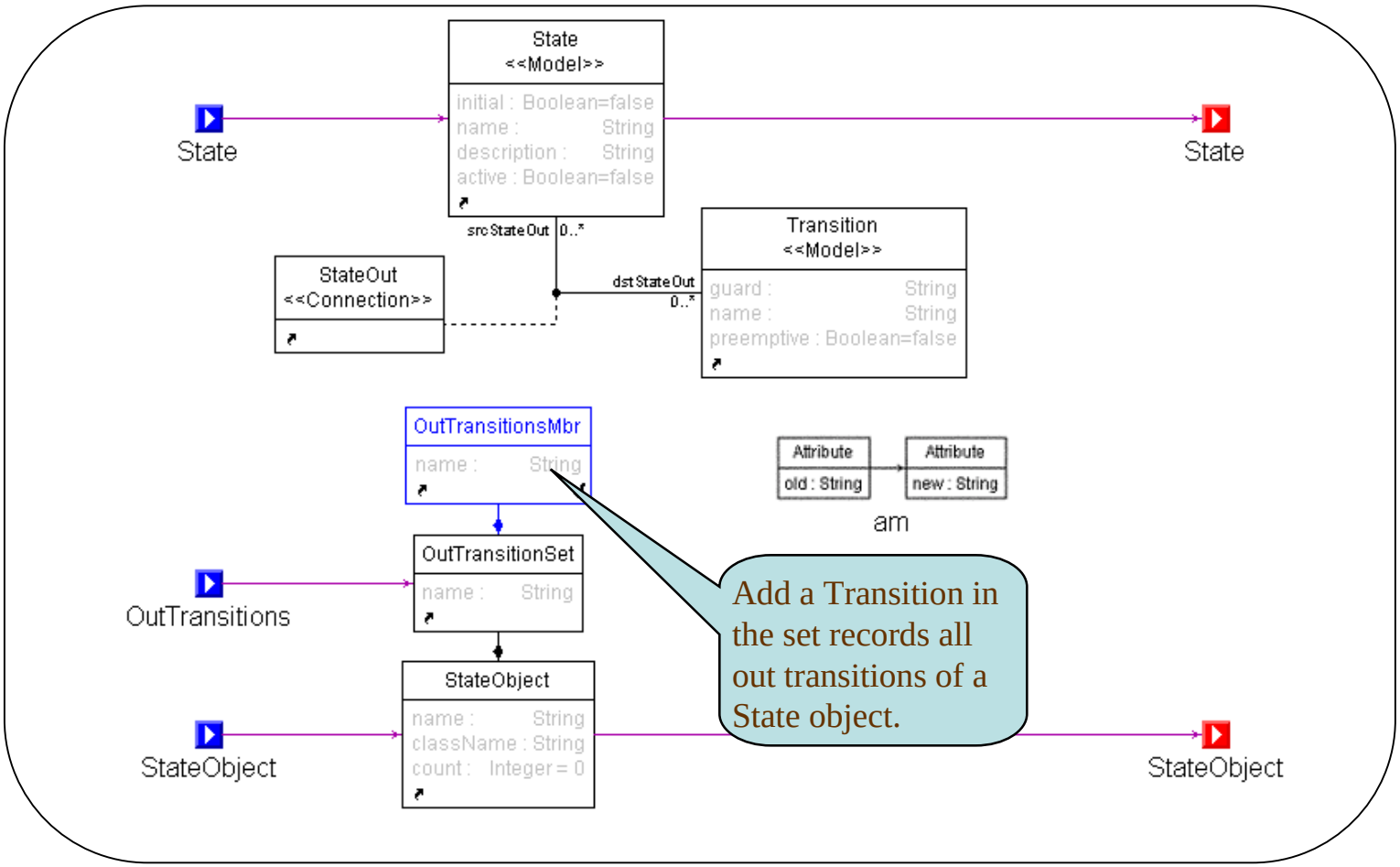
- The semantic mapping specification for FML consists of a sequence of mapping rules.



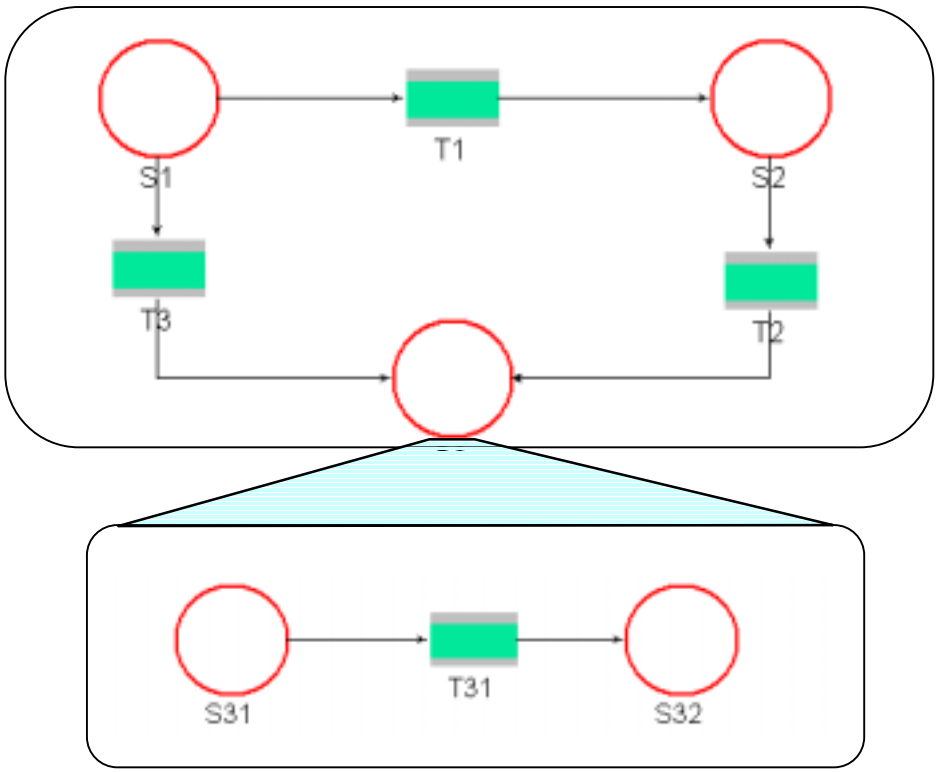
Mapping Rule : Create State Objects

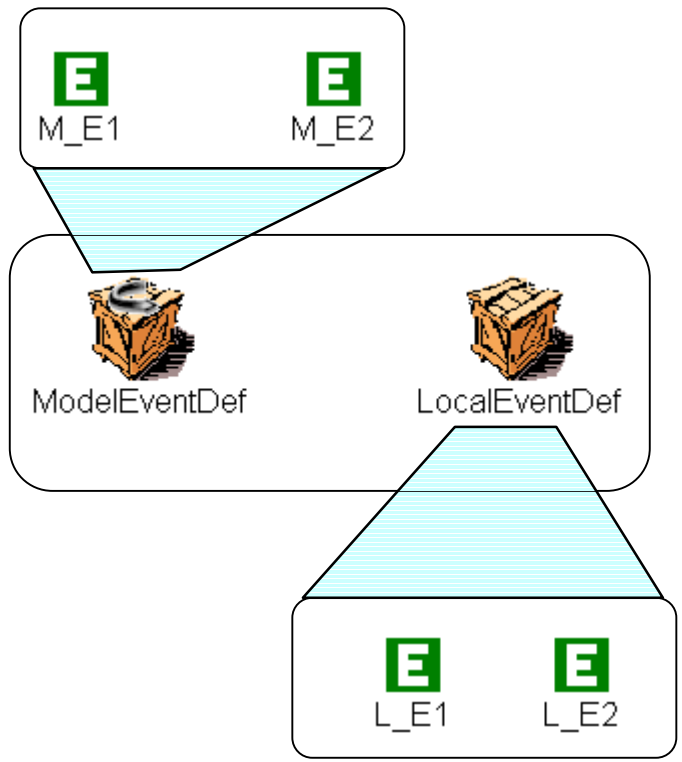


Mapping Rule : Set Out Transition Values

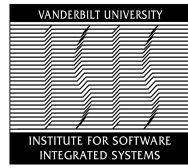


A Hierarchical FSM Model



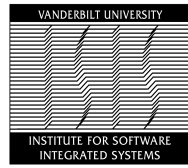


Output XML file



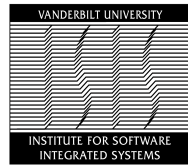

```
- <AsmLDataFile _id="id671" count="19" filename="" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="C:\research\semantics\new\UDM\ASMSfw.xsd">
  - <AsmLObject _id="id7ab" idx="6" name="FSM_Example" count="5" className="FSM">
      <AttributeVal _id="id7c0" idx="0" name="name" value="FSM_Example" />
      <AttributeVal _id="id7ed" idx="3" name="initialState" value="S1" />
      <SequenceVal _id="id7d4" idx="1" name="outputEvents" />
      <SetVal _id="id7df" idx="2" name="localEvents" />
    - <SetVal _id="id7fd" idx="4" name="children">
        <SetMbr _id="id808" name="S1" />
        <SetMbr _id="id80a" name="S2" />
        <SetMbr _id="id80b" name="S3" />
      </SetVal>
    </AsmLObject>
  + <AsmLObject _id="id82b" idx="7" name="S1" count="8" className="State">
  + <AsmLObject _id="id82c" idx="8" name="S2" count="8" className="State">
  - <AsmLObject _id="id82d" idx="9" name="S3" count="8" className="State">
      <AttributeVal _id="id845" idx="3" name="inital" value="false" />
      <AttributeVal _id="id846" idx="2" name="active" value="false" />
      <AttributeVal _id="id847" idx="0" name="name" value="S3" />
      <AttributeVal _id="id855" idx="4" name="initialState" value="S31" />
      <AttributeVal _id="id865" idx="5" name="master" value="null" />
    - <SetVal _id="id875" idx="6" name="slaves">
        <SetMbr _id="id880" name="S32" />
        <SetMbr _id="id881" name="S31" />
      </SetVal>
    - <SetVal _id="id892" idx="7" name="outTransitions">
        <SetMbr _id="id8a4" name="T3" />
      </SetVal>
    </AsmLObject>
```

Output from the specification generator



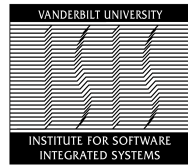

```
structure ModelEvent implements Event
  case M_E1
  case M_E2
  case M_E3
structure LocalEvent implements Event
  case L_E1
  case L_E2
  case L_E3
var FSM = new FSM ([], {}, S1, {S1,S2,S3})
var S1 = new State (false, true, null, null, {}, {T1})
var S2 = new State (false, false, null, null, {},{T2})
var S3 = new State (false, false, S31, null, {S31, S32}, {T3})
var S31 = new State (false, true, null, S3, {}, {T31})
var S32 = new State (false, false, null, S3, {}, {})
var T1 = new Transition (true, false, ModelEvent.ME2, ModelEvent.ME1, S1, S2)
var T2 = new Transition (true, true, ModelEvent.ME3, ModelEvent.ME2, S2, S3)
var T3 = new Transition (true, false, ModelEvent.ME1, ModelEvent.ME2, S3, S1)
var T31 = new Transition (true, false, ModelEvent.ME3, LocalEvent.LE1, S31, S32)
```

Conclusions



- These is no formalized design process for the DSML.
  - Solution : a systematic DSML design framework.
- The semantics definition for the DSML is weak.
  - Solution : formal semantic domain specification & semantic mapping specification.
- The “domain-specific” limits the reusability of the DSML.
  - Solution : MoC & Semantic Anchoring.

Future Work



- MoC semantics unites definition
- DSML composition
  - Metamodel composition
  - Semantics composition