Analysis of a Metamodel to Estimate Complexity of Using a Domain-Specific Language

Jonathan Sprinkle

The University of Arizona

ece

This work is supported by AFOSR under award #FA9550-091-0519, and NSF under awards CNS-0915010 and CNS-0930919.

You might have seen this before...

T.J. McCabe. "A Complexity Measure". Software Engineering, IEEE Transactions on, SE-2(4):308-320.

308 IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-2, NO. 4, DECEMBER 1976

A Complexity Measure

THOMAS J. McCABE

Abstract—This paper describes a graph-theoretic complexity measure and illustrates how it can be used to manage and control program complexity. The paper first explains how the graph-theory concepts apply and gives an intuitive explanation of the graph concepts in programming terms. The control graphs of several actual Fortran programs are then presented to illustrate the correlation between intuitive complexity and the graph-theoretic complexity. Several properties of the graph-

in conjunction with the complexity measure; a testing strategy is defined that dictates that a program can either admit of a certain minimal testing level or the program can be structurally reduced.

Index Terms—Basis, complexity measure, control flow, decomposition, graph theory, independence, linear, modularization, programming, reduction, software, testing.

I. Introduction

THERE is a critical question facing software engineering today: How to modularize a software system so the resulting modules are both testable and maintainable? That the issues of testability and maintainability are important is borne out by the fact that we often spend half of the development time in testing [2] and can spend most of our dollars maintaining systems [3]. What is needed is a mathematical technique that will provide a quantitative basis for

tified and analyzed by the tools described in this paper.

Manuscript received April 10, 1976.
The author is with the Department of Defense, National Security Agency, Ft. Meade, MD 20755.

II. A Complexity Measure

In this section a mathematical technique for program modularization will be developed. A few definitions and theorems from graph theory will be needed, but several examples will be presented in order to illustrate the applications of the technique.

The complexity measure approach we will take is to measure and control the number of paths through a program. This approach, however, immediately raises the following nasty problem: "Any program with a backward branch potentially has an infinite number of paths." Although it is possible to define a set of algebraic expressions that give the total number of possible paths through a (structured) program,[1] using the total number of paths has been found to be impractical. Because of this the complexity measure developed here is defined in terms of basic paths—that when taken in combination will generate every possible path.

The following mathematical preliminaries will be needed, all of which can be found in Berge [1].

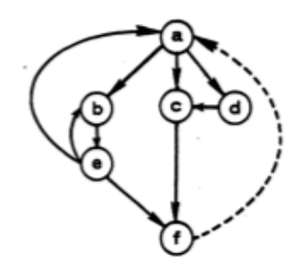
Definition 1: The cyclomatic number V(G) of a graph G with n vertices, e edges, and p connected components is

$$v(G) = e - n + p.$$

Theorem 1: In a strongly connected graph G, the cyclomatic number is equal to the maximum number of linearly independent circuits.

The applications of the above theorem will be made as follows: Given a program we will associate with it a directed graph that has unique entry and exit nodes. Each node in the graph corresponds to a block of code in the program where the flow is sequential and the arcs correspond to branches taken in the program. This graph is classically known as the program control graph (see Ledgard [6]) and it is assumed that each node can be reached by the entry node and each node can reach the exit node. For example, the following is a program control graph with entry node "a" and exit node "f."

G:



[1] See the Appendix.

terms. The control graphs of several actual Fortran programs are then presented to illustrate the correlation between intuitive complexity and the graph-theoretic complexity. Several properties of the graph-

Briefly:

$$v(G) = e - n + 2$$

Manuscript received April 10, 1976.
The author is with the Department of Defense, National Security Agency, Ft. Meade, MD 20755.

Karsten Ehrig, Jochen Malte Küster, Gabriele Taentzer. "Generating instance models from meta models" *Softw. Syst. Model.* (2009) 8:479–500.

Briefly:

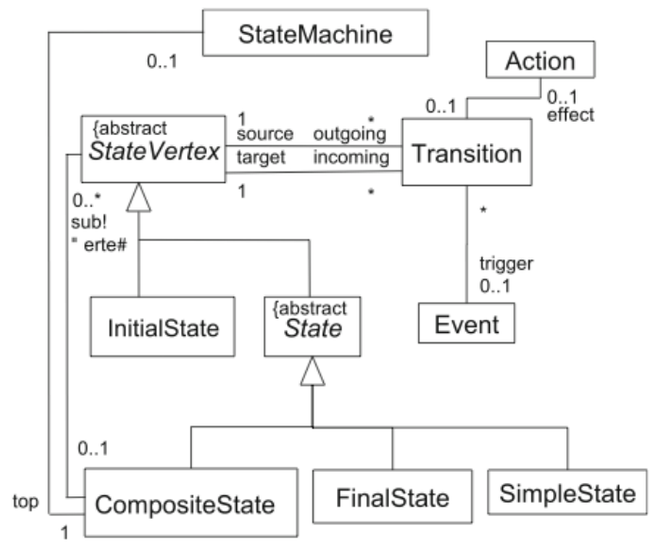
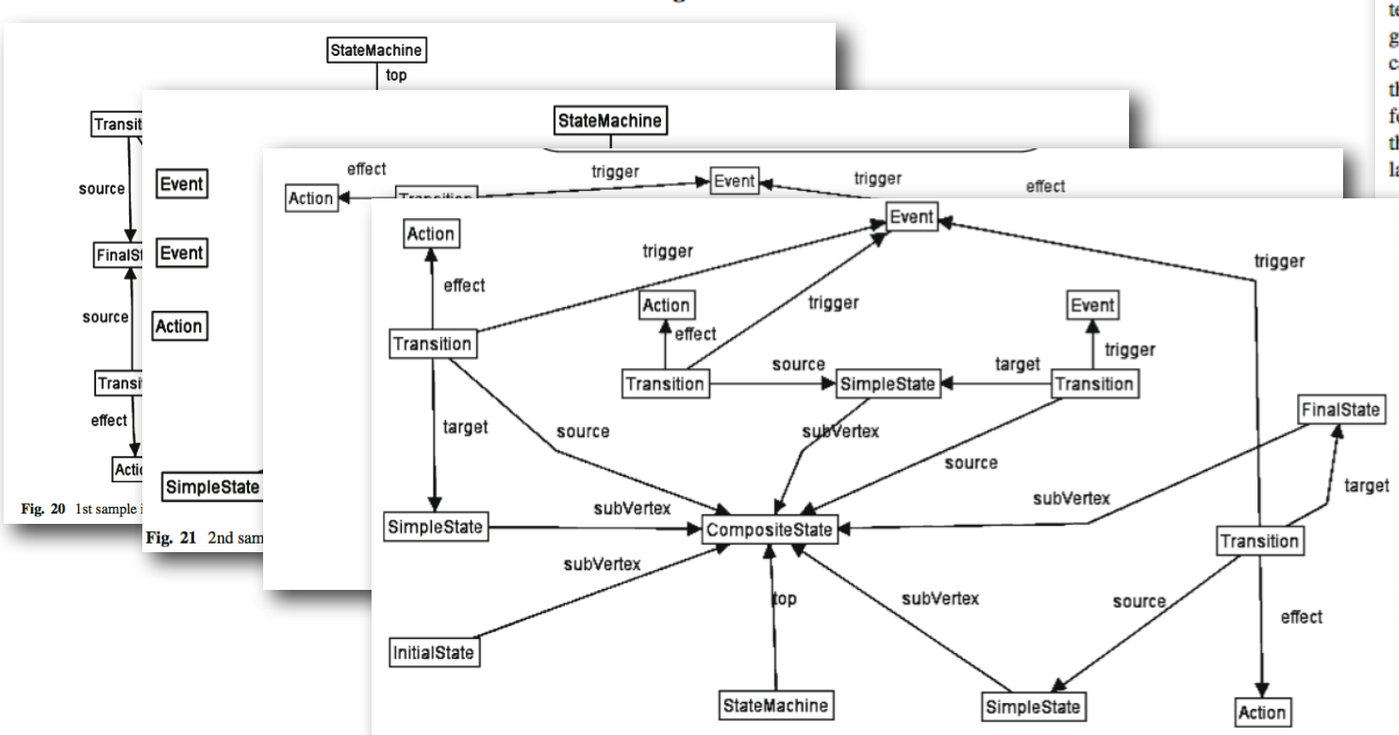


Fig. 1 Meta model for statecharts



Softw Syst Model (2009) 8:479–500
DOI 10.1007/s10270-008-0095-y

THEME SECTION

Generating instance models from meta models

Karsten Ehrig · Jochen Malte Küster · Gabriele Taentzer

Received: 1 April 2007 / Revised: 9 May 2008 / Accepted: 16 May 2008 / Published online: 1 July 2008
© Springer-Verlag 2008

Abstract Meta modeling is a wide-spread technique to define visual languages, with the UML being the most prominent one. Despite several advantages of meta modeling such as ease of use, the meta modeling approach has one disadvantage: it is not constructive, i.e., it does not offer a direct means of generating instances of the language. This disadvantage poses a severe limitation for certain applications. For example, when developing model transformations, it is desirable to have enough valid instance models available for large-scale testing. Producing such a large set by hand is tedious. In the related problem of compiler testing, a string grammar together with a simple generation algorithm is typically used to produce words of the language automatically. In this paper, we introduce instance-generating graph grammars for creating instances of meta models, thereby overcoming the main deficit of the meta modeling approach for defining languages.

eywords Meta model · UML · Graph grammar · Instance eneration

ommunicated by Prof. Thomas Kühne.

. Ehrig (✉)
epartment of Computer Science, University of Leicester, eicester, UK
mail: karsten@mcs.le.ac.uk

. M. Küster
BM Zurich Research Laboratory, 8803 Rüschlikon, Switzerland
mail: jku@zurich.ibm.com

. Taentzer
epartment of Computer Science, Philipps-University, larburg, Germany
mail: taentzer@mathematik.uni-marburg.de

1 Introduction

With models expressed in the Unified Modeling Language (UML) [35] becoming widely used in software engineering, also the meta modeling approach to define the syntax of modeling languages has gained a wide acceptance: Commonly, a meta model is designed which defines the abstract syntax of the language in a declarative way. Instantiation of the meta model then yields a concrete model.

The meta modeling approach has several advantages, one of them being that a visual meta model allows a quick grasp of the concepts being defined. Further, the meta modeling approach is also beneficial when it comes to defining complex modeling languages, consisting of several individual models. Nevertheless, there exists also one disadvantage: Whereas constructing words of a language defined by a string grammar can easily be done by applying grammar derivations, meta model instantiation is hard to operationalize, due to the declarative form of a meta model.

In common applications of the UML, this does not pose a problem because the process of instantiation is performed by the software engineer when constructing models. However, there are a number of emerging applications where firstly an approach is needed for generating a large set of instances automatically and secondly the generation process must also be described explicitly. In other words, instead of the declarative meta model an equivalent operational description of the language defined by the meta model is required.

Important applications of an operational description of the language defined by a meta model include automated testing of model transformations and code generators as well as testing of model debuggers [27]. Model driven architecture [34] favors a widespread usage of model transformations. The quality of model transformations is thus crucial for their successful and widespread usage and needs to be validated by

Springer

- *Execution paths*
- *Constraints on execution*
- *End-users who have problems*

- DSM aims to (among many possibilities):
  - Raise the level of abstraction
    - introducing domain-specific types
  - Improve expressivity
    - but not at the limit (i.e., a single-concept programming language)
  - Restrict the programmer/modeler
    - introducing dependencies on creating certain concepts
  - Enable a “correct by construction” approach
- This suggests a few potential metrics that can be *applied to a metamodel which is used to generate a language*
  - Given a metamodel, how hard is it to create an instance of every potential object in the metamodel?
  - Given a metamodel, what is the minimum set of elements required to create a syntactically valid model?
  - Given a modeling language (metamodel, plus semantic mapping) what is the minimum set of elements required to create a semantically valid model?
  - Given a modeling language, what is the complexity of creating a new syntactically and semantically valid model?

- In this paper we focus on the first element:
  - Given a metamodel, how hard is it to create an instance of every potential object in the metamodel?
- Assumption:
  - An algorithmic metric for a domain-specific language is useful
- Notas bene:
  - There is no claim that the proposed is a sufficient (or even necessary) metric for a language
  - Complementary metrics for notation (including metrics for visual languages) should be considered alongside those in this paper
    - Usability, for example, may have different metrics that depend on concrete syntax, tool layout, etc.
- Potential impact:
  - Examine a metamodel, tell the language designer a single number that may surprise them

- There is significant work in *model* metrics
  - Metrics that gauge the complexity of *constructed models*
- This paper is orthogonal to any work in model metrics
- Further, the complexity of a metamodel is *not necessarily* tied to the complexity of a language for which it describes the abstract syntax*

* This is unlikely to provoke an argument with anyone attending the workshop.

- Eessaar's work on applying the *constructed model* metrics of a language to the models created by a similar language
- Creating *constructed model* metrics based on domain-specific concepts, gathered from the metamodel (McQuillan and Power)
- Metrics to consider the descriptive capability of a language, technique, or method, based on the metamodel (of the development method) by Rossi and Brinkkemper*

* M. Rossi, S. Brinkkemper. "Complexity Metrics for Systems Development Methods and Techniques." *Information Systems*, 21(2):209-227, 1996.

- Create an analog to the McCabe cycles and graphs, which creates a state model from the language metamodel
- Utilize this graph as the generating structure for any complexity metrics

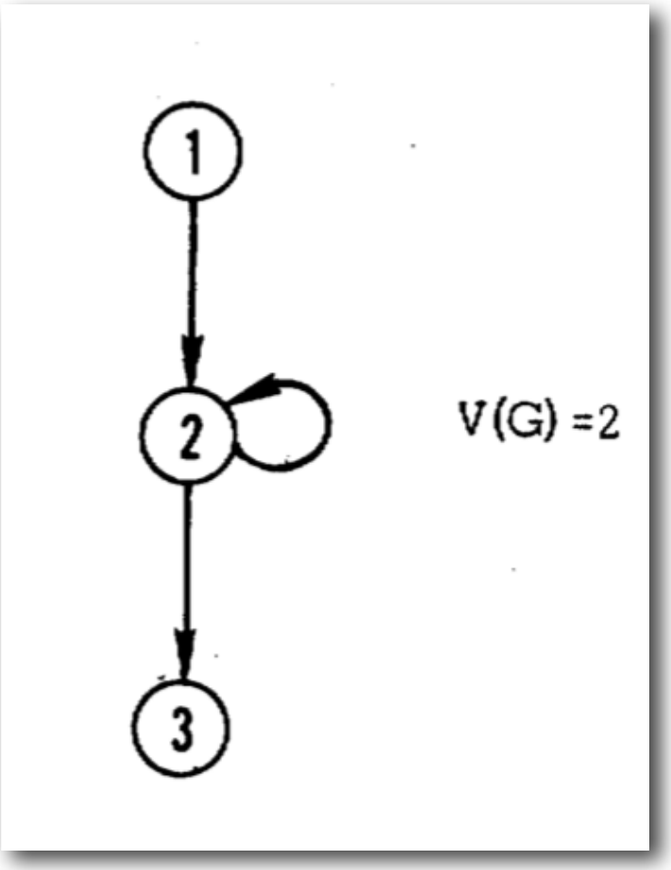
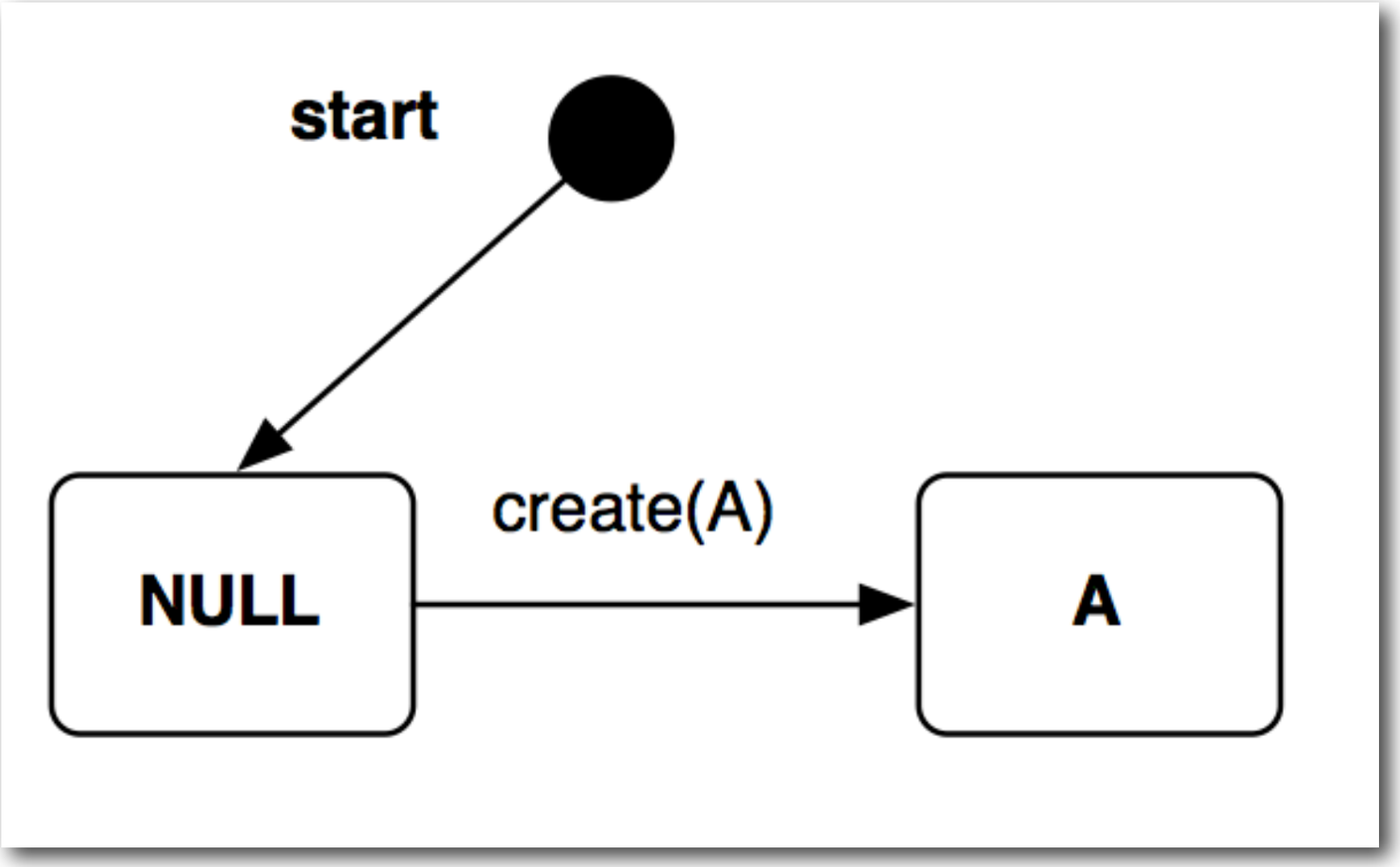
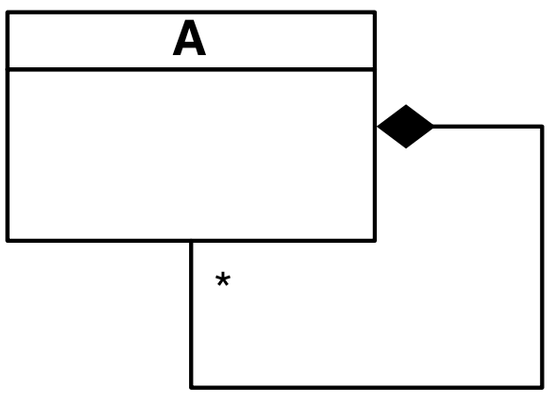


Fig. from McCabe

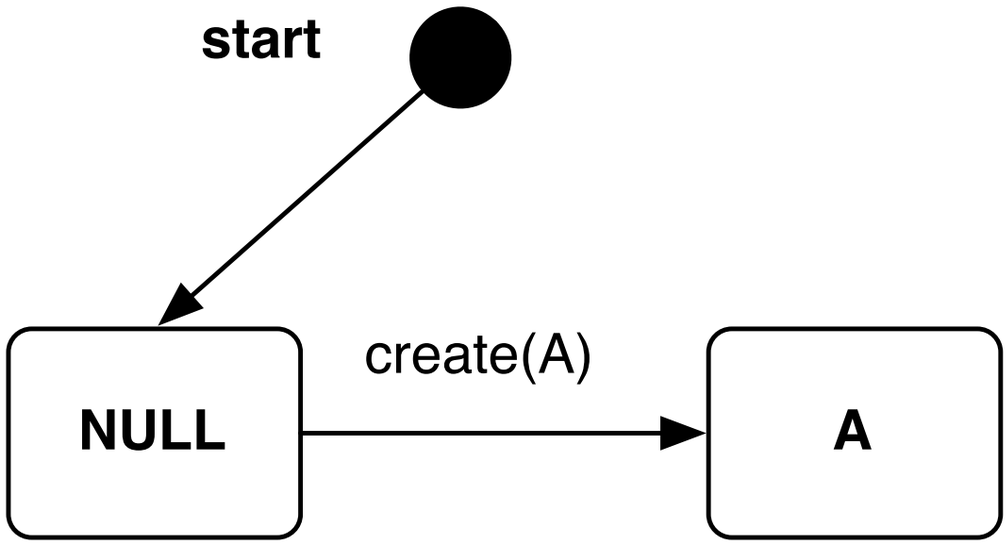


Proposed Style

- What metamodeling elements are considered?
  - Types
  - Containment
  - Inheritance
  - Association
- Not (yet) covered in the algorithms:
  - Reference/pointers
  - N-ary associations or aggregation association (e.g., <<Set>> in GME)
  - Aspects/visualization
  - Attribute value setting/insetting
  - Composition and association constraints from the diagram
  - General OCL constraints
  - Module interconnection (component association between ports)

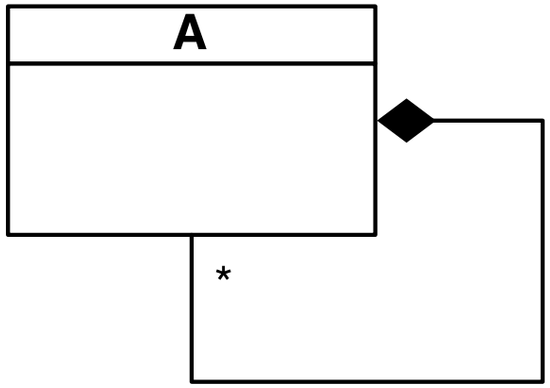


Metamodel

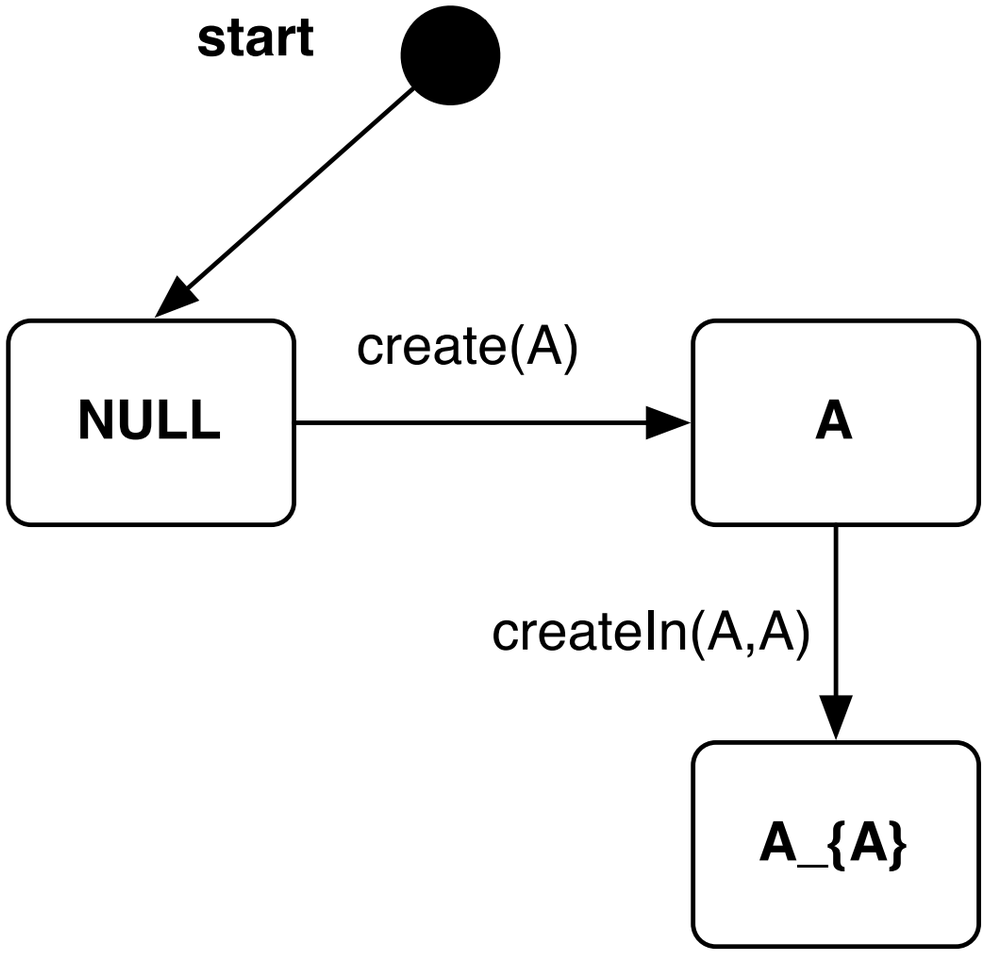


Creates at least one 'A' model

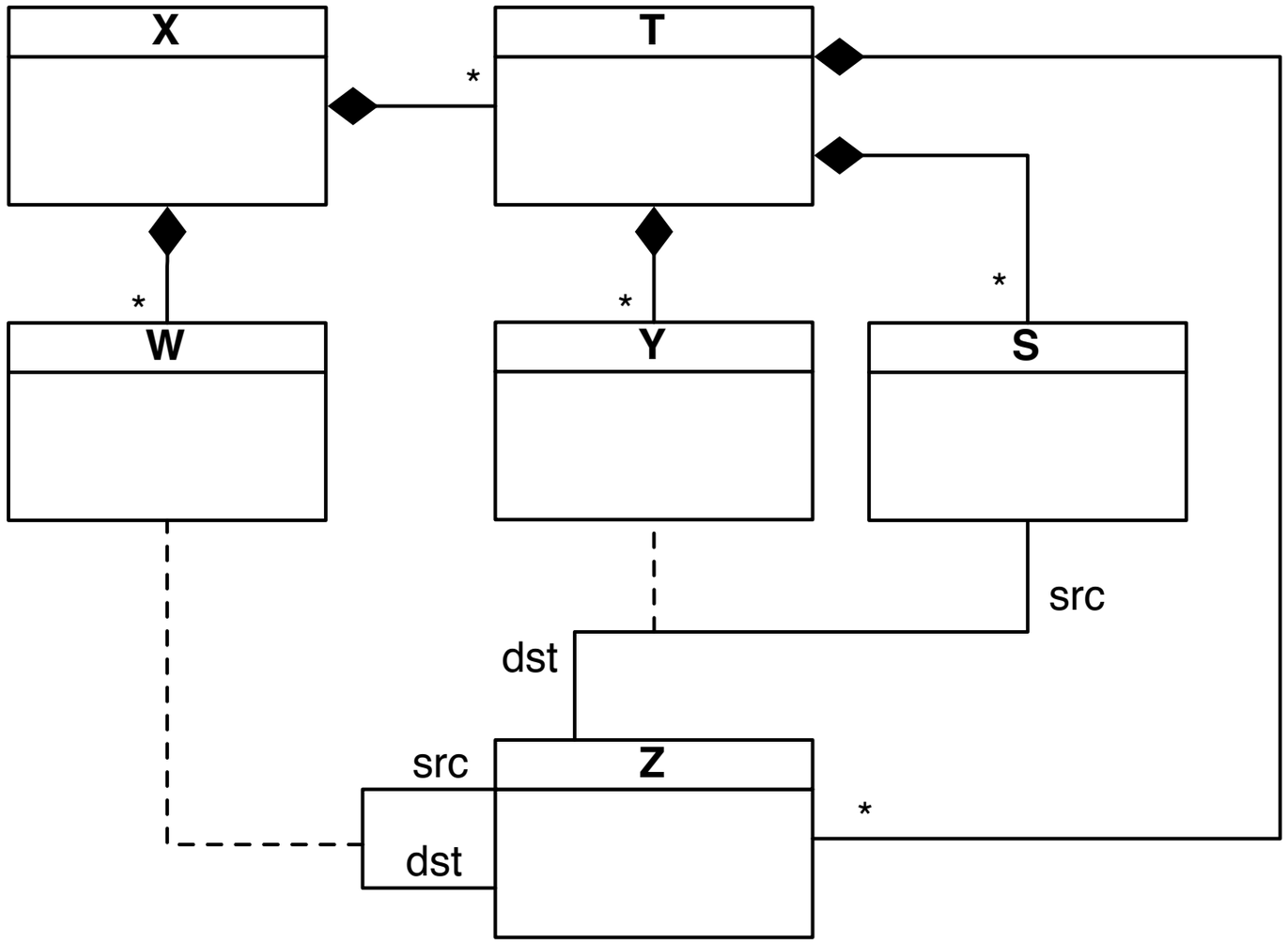
Missing: The containment association that permits hierarchical composition of 'A'

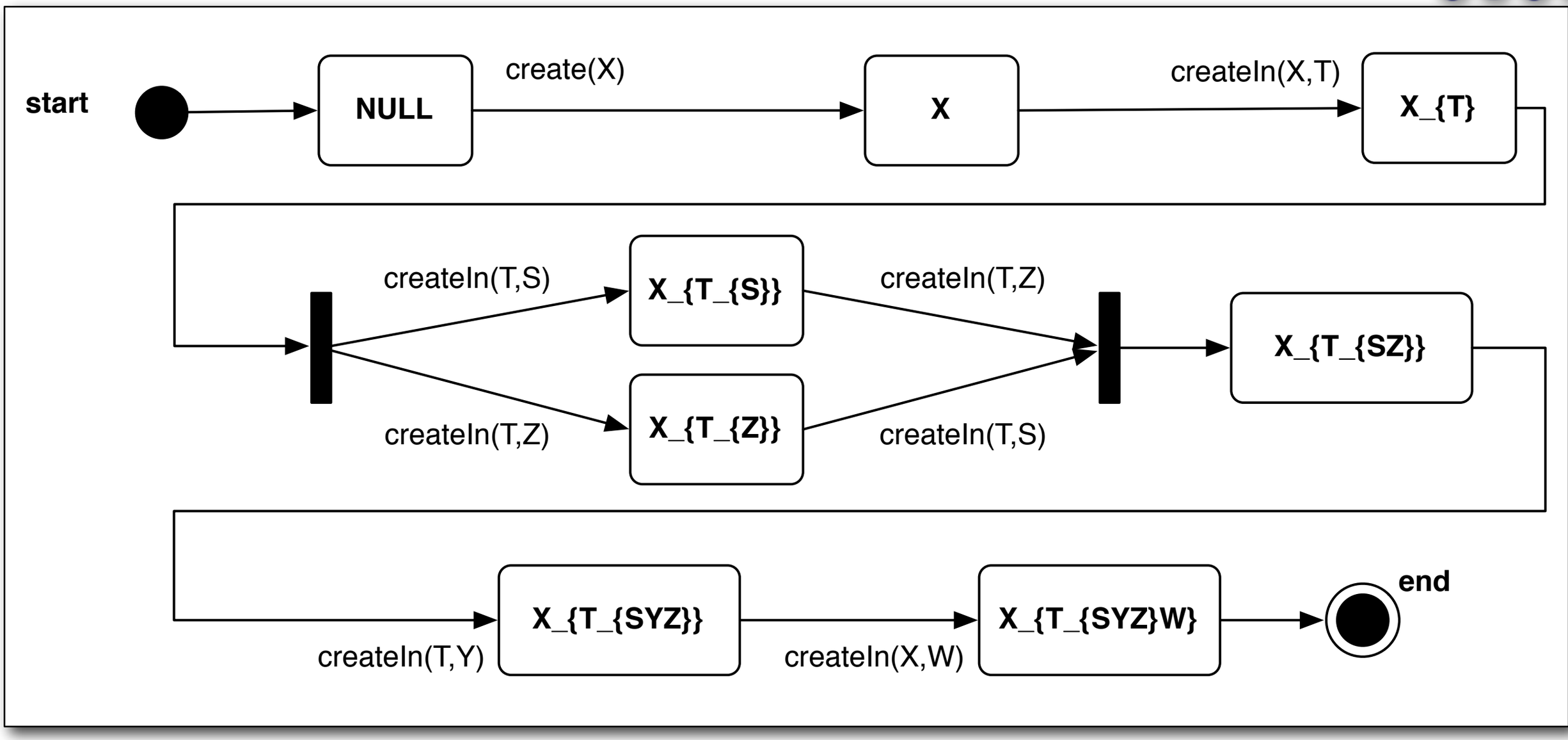


Metamodel



Creates at least one 'A' model, and exercises the containment association


X
T
*
*
*
*
W
Y
S
src
dst
src
Z
*
dst



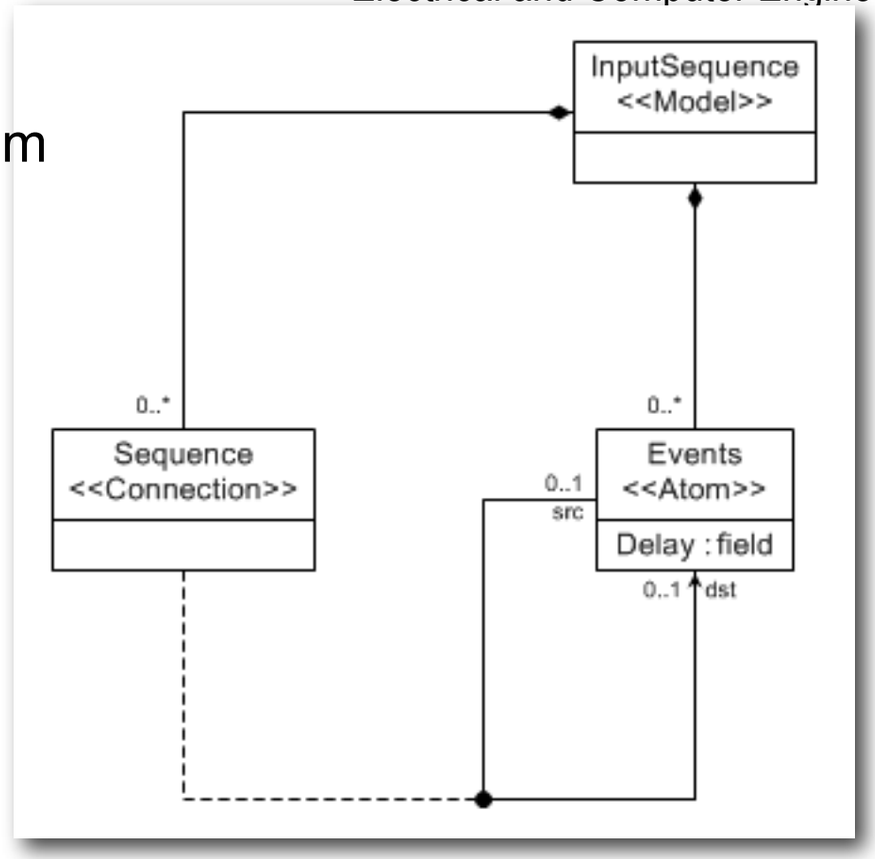
Briefly:

$$\emptyset, X, X_T, \{X_{T_S}, X_{T_Z}\}, X_{T_{SZ}}, X_{T_{SYZ}}, X_{T_{SYZ}W}$$

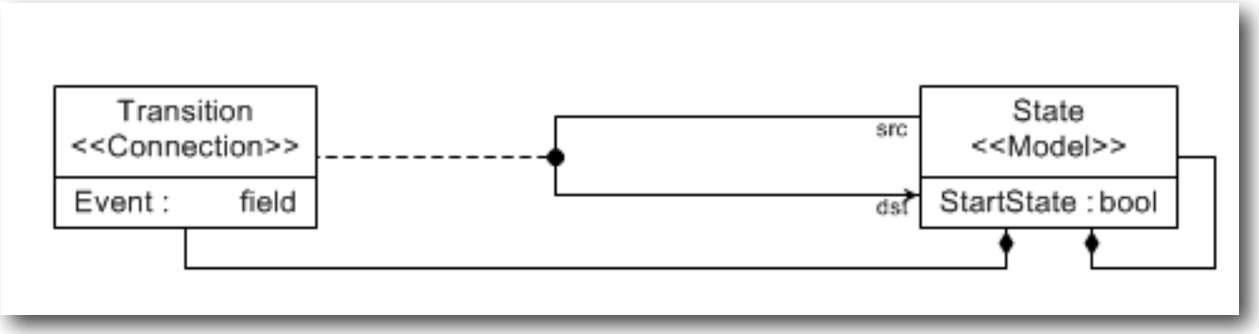
Briefly-er:

$$\emptyset, X, (T, (\{S, Z\}, SZ, SYZ), W)$$

Hierarchical Finite State Models (HFSM) Paradigm
From GME10, Available:
http://www.isis.vanderbilt.edu/projects/gme/

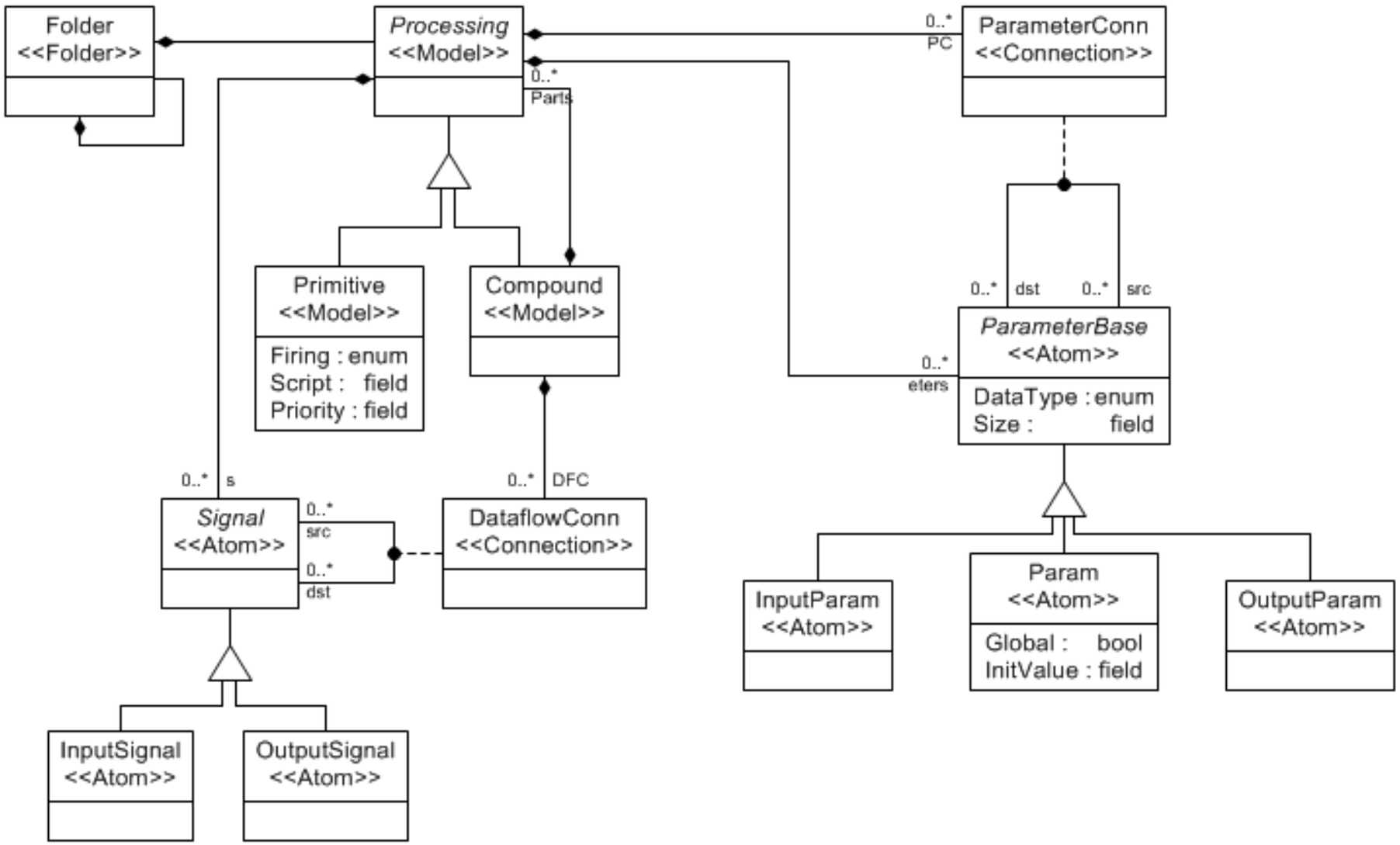


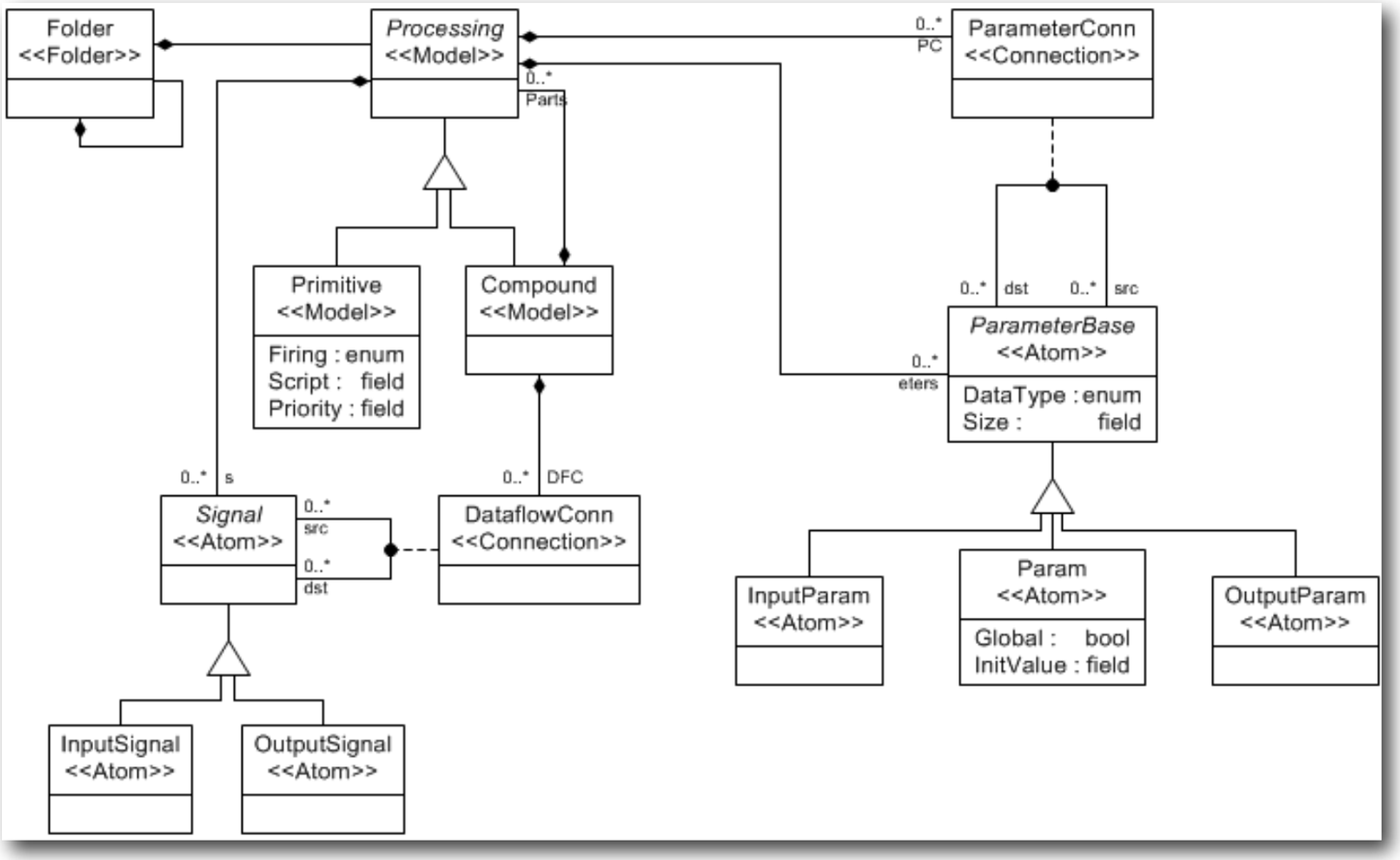
Note that there is a fork in creating either a **State** or an **InputSequence**, but that creating (e.g.) a **State** inside the existing state *must* occur prior to creating the **Transition**, because that transition requires a **State** model to participate in the source and destination of that association.



$$\emptyset,$$
$$\{State, (State, Transition),$$
$$InputSequence, (Events, Sequence)\}$$

Signal Flow (SF) Paradigm
From GME10, Available:
http://www.isis.vanderbilt.edu/projects/gme/





$$\emptyset, Folder, (\{Folder, Compound, (\{Compound, InputParam, Param, OutputParam, InputSignal, OutputSignal, \{DataflowConn, ParameterConn\}\}, Primitive\}).$$

- Complexity of Exercise
  - The number of instantiation *elements* that must be done in *order (O)*, minus the number of instantiation elements that can be done in any order *(C)*, for "choice"

$$C_e = O - C$$

- $C_e(\text{HFSM})$ =6, is calculated as follows:
  - O=8
    - **State**, (**State**, **Transition**) (2)
    - **State**, **Transition** (2)
    - **InputSequence**, (**Events**, **Sequence**) (2)
    - **Events**, **Sequence** (2)
  - C=2
    - **State** (1)
    - **InputSequence** (1)

Analogous to McCabe's linearly independent paths.

- **Folder**, (...) (2)
- **Folder**, **Compound**, (...), **Primitive** (4)

- Similarly, $C_e(\text{SF})$ =4, is calculated using O=6, C=2

- **Compound**, **InputParam**, **Param**, **OutputParam**, **putSignal**, **OutputSignal** (1)
- **DataflowConn**, **ParameterConn** (1)

- Expansion of Cardinality
  - Number of elements in the metamodel whose final cardinality is greater than 1, when counted over the entire metamodel
- $E_c(\text{HFSM}) = 1$
  - State must be created twice
- $E_c(\text{SF}) = 2$
  - Folder and Compound must be created twice

Analogous to expressivity (the lower, the better).

- Note that despite its analogy to expressivity, here in the limit, improvement does not reduce or increase the types in the metamodel, but rather considers issues of cardinality and association, and is independent of inheritance and number of types.

- Not a final word, but a start on domain-independent metrics for domain-specific metamodels
- Provides an analog to the high-impact work of McCabe
- Provides an analog to expressivity that does not compromise DSM goals
- Future Work
  - Incorporate constraints (through guards on transitions between states)
  - A natural analog, but now requires extensions to McCabe's unguarded transitions
  - However, easily permits transition constraints that involve attributes, etc.
  - Considerations of the necessary minimum set required in order to have a semantically meaningful model, rather than a fully exercised set.

This work is supported by AFOSR under award #FA9550-091-0519, and NSF under awards CNS-0915010 and CNS-0930919. The author is grateful to the anonymous reviewers for their kind and insightful comments that led to improvements in the final version of this paper.

THE UNIVERSITY OF ARIZONA

We are always looking for good graduate students.

http://ece.arizona.edu/