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

Jonathan Sprinkle

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

Briefly:

\[ v(G) = e - n + 2 \]

Briefly:

Abstract: Meta modeling is a widespread 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.

Keywords: Meta model - UML - Graph grammar - Instance generation
What do these papers have in common?

- **Execution paths**
- **Constraints on execution**
- **End-users who have problems**
How to apply to DSM?

- 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?
What is covered in this paper?

• 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
(Semi-)related Work

- 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.
Related Work

- 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*

---

Approach

- 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
Approach

- 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)
Start with an example

Metamodel

Missing: The containment association that permits hierarchical composition of ‘A’
Start with an example

Metamodel

Create at least one ‘A’ model, and exercises the containment association
A more complex example
Briefly:
\[
\emptyset, X, X_T, \{X_{TS}, X_{TZ}\}, X_{TSZ}, X_{TSYZ}, X_{TSYZW}
\]

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.

$$\emptyset,$$

$$\{\text{State, (State, Transition),}$$

$$\text{InputSequence, (Events, Sequence)}\}$$
Signal Flow (SF) Paradigm
From GME10, Available:
http://www.isis.vanderbilt.edu/projects/gme/
Some actually well-established metamodels, next: SF (p2)

∅, Folder,
({Folder, Compound, InputParam, Param, OutputParam, InputSignal, OutputSignal, {DataflowConn, ParameterConn} }),
Primitive
Rudimentary Metric: Complexity of Exercise

- **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”

$$C_e = O - C$$

- $C_e(HFSM) = 6$, 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)

- Similarly, $C_e(SF) = 4$, is calculated using $O = 6$, $C = 2$:
  - Folder, (...), Primitve (4)
    - Folder, Compound, (...), Primitive (4)
    - Compound, InputParam, Param, OutputParam, putSignal, OutputSignal (1)
    - DataflowConn, ParameterConn (1)

Analogous to McCabe’s linearly independent paths.
Rudimentary Metric: Expansion of Cardinality

- 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.
Conclusions

• 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.
We are always looking for good graduate students.

http://ece.arizona.edu/