The Value of Domain Specific Languages

Steve Hickman

OMG Model Integrated Computing Workshop

October 2004

Outline

- Define domain specific and language
- Requirements all languages must meet
- Illustrate the domains of various languages
- Show how and where domain boundaries intersect
- Display examples of how language choices can impact productivity
- Explain why this approach is more productive than attempting to use a single language
- Detail what modeling tools must do to accommodate this approach.

Definitions

- Languages are defined by their verbs.
  - Noun: A label for a concept or set of concepts.
  - Verb: An action on a noun or nouns. Verbs can be treated as nouns (because they define a thing – in this case an action)
  - Neither of these requires the Roman alphabet be part of the symbology.
- Domain:
  - A well bounded coherent set of meta-concepts
- Consequences:
  - All languages are domain specific; the only question is what the domain boundaries are.
- Languages are defined by the concepts they can represent as well as the concepts they emphasize.

Requirements of All Languages

- Well defined syntax
- Well defined semantics
  - Users need it to understand and properly use a language.
  - Tool implementers need it to insure consistency across companies and across time.
  - Without well defined semantics, it becomes difficult or impossible to correctly map concepts from one representation to another or one language to another.
  - Without well defined semantics, language evolution becomes problematic.
  - Well defined semantics does NOT mean just a list of example behaviors. A semantics that is not bounded leaves holes for goblins.

Example Languages

- Programming languages such as C, C++, Java can represent “computable”/ “Turing complete” concepts they do not directly emphasize data flow dependencies. (This can be partially derived from the representation in C, C++, Java, but because it is derived it might not be as optimal as it would have been had the concept originally been modeled using a language that emphasized data flow).
- C++, Java and C# also more succinctly represent concepts related to event driven and polymorphic behavior (because of their object oriented nature) than older languages like C or FORTRAN.
- Aspect-Oriented languages handle additional concepts

But what about “Modeling Languages”?

- Data flow diagrams
  - Can represent both continuous and discrete computation.
- Some languages, such as Class Diagrams or rule-based languages, simply cannot represent certain concepts:
  - Class Diagrams primarily represent static structure.
  - (Pure) Rule-based languages have no looping construct so they aren’t Turing complete like traditional languages.
  - Older rule based languages cannot represent time.

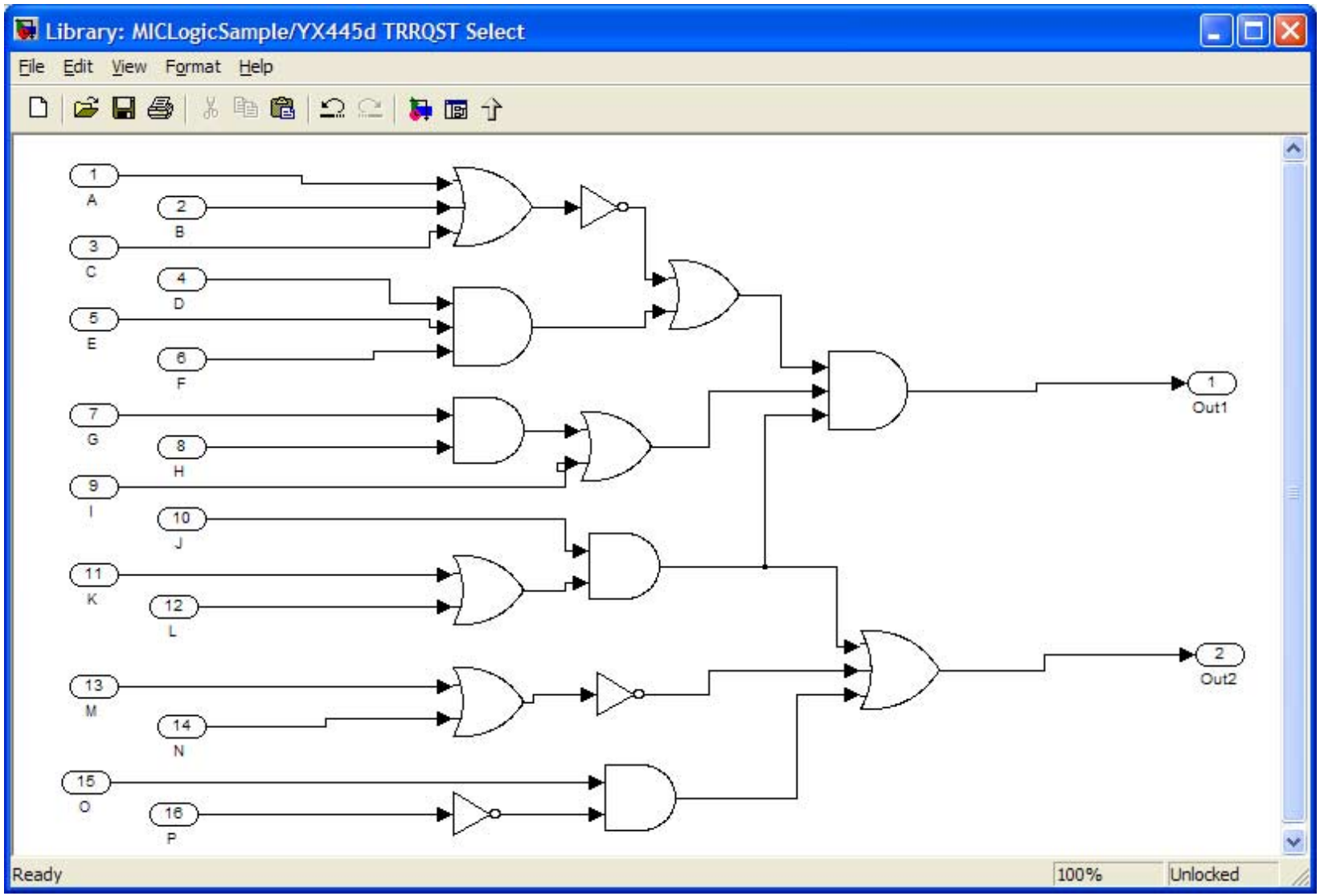
UML isn’t a Language?

- UML is a collection of languages that have fairly well defined mutual interfaces. Examples:
  - UML State Machines:
    - State Machines
  - UML Sequence Diagrams:
    - Sequential interaction
  - UML Class Diagrams
    - Static structure

Concept vs. Symbology

- Need to make a distinction between the concepts a language can represent and the symbology / visual representation used to represent those concepts.
- Are these two following visual representations the same expression or are they different expressions?

Logical Expression – Data Flow



Logical Expression - Text

- OUT1= (((NOT (A OR B OR C)) OR (D AND E AND F)) AND ((G AND H) OR I) AND (J AND (K OR L)))
  - Infix notation
- OUT2 = (OR (AND J (OR K L)) (NOT (OR M N)) (AND O (NOT P)))
  - Prefix notation

Same or Different?

- Answer: It all depends – because we as humans assign semantics to the representation.
- Issues such as operator precedence, short-circuit semantics and the models of computation affect the determination.

Domain Interfaces vs. Intersections

- Intersection
  - Prior example: All of logical expressions.
  - UML 1.x – Sequence Diagrams & Object Interaction diagrams
  - UML 1.x – State Diagrams & Activity Diagrams.

Domain Interfaces vs. Intersections

- Interfaces - UML Languages
  - Sequence Diagrams:
    - Each execution occurrence on a LL can correspond to an Activity Diagram
    - At a requirements level each requirement can correspond to a single Activity or Sequence Diagram
  - State Diagrams
    - Each transition can correspond to an Activity Diagram / Sequence Diagram

Domain Interfaces vs. Intersections

- Interfaces in Matlab
  - Simulink <-> Stateflow: Events/state variables at a stateflow block boundary
- Programming Languages
  - Keep in mind we aren't referring to application interfaces. Languages that can express the same concepts merely describe different ways to represent those concepts.

Where Interfaces Occur

- Language interfaces are defined by "*equating*" nouns. In effect what we are doing is telling others that the things in question have some relationship. When we want to view the thing from a particular aspect/ domain, we use the associated representation.
- Equating can mean:
  - "Equivalence"
  - "Another aspect"
  - "Class-instance"

Where Intersections Occur

- Language intersections are equivalent to subgraph isomorphisms, only in this case both nodes (nouns) and edges (verbs) have uniquely identifying characteristics. All relevant characteristics must be identical, not merely similar.
- Keep in mind this occurs at the language/ meta level – not at the instance level. In the prior example, the intersection occurs because we equate particular graphical symbols with corresponding ASCII symbols.

Language and Productivity

- Productivity: The reason modeling exists
- Utility has changed
  - Used to be abstract communication only
  - Now includes such things as simulation, code and test generation.

Which is more productive?

- Need to be clear about the distinction between language and symbology/ representation
- In this example, logical expressions can be viewed as a common subset of two languages that use different symbology.
- Could also be viewed as a simple (sub) language with multiple representations.

So which is more productive?

- During creation or communication?
- Creation
  - Textual: 91 chars (infix) or 61 chars (prefix)
  - Graphical: keystrokes/ mouse moves depend on tool and user familiarity.
- Communication
  - Textual: parsing can be difficult
  - Graphical: data flow, dependencies more obvious

Why multiple languages are more productive

- Each language can emphasize different things:
  - Activity Diagrams emphasize data flow / natural parallelism
  - Sequence Diagrams emphasize order of interaction
  - Class Diagrams emphasize static structure

Productivity using Multiple Languages

- Simple, compact tools keep focus on relevant issues
- Goal of modeling is to “simplify”. The simplification must be from the perspective of the user, not the tool creator.
  - Multiple representations allow creator and user to view same model differently

The Main Point

- One size does NOT and need not fit all
- Multiple languages allow customization that focuses the domain appropriately
- Language is about communication. In order to properly communicate, language semantics must be well defined.
- Because some problems are not confined to a single domain, there will be a need to define interfaces between different domains.
- OMG has defined a set of metamodels (UML) that handle the most commonly defined concepts.
- OMG has also defined MOF to enable the creation of new metamodels => new languages.

What Tools Must Support

- Enable specification in one symbology and viewing in another, whether or not this involves one or multiple languages.
- Easy mapping of one language construct to another. This is the issue of “language interfaces”. Tools must be able to make distinctions between these different kinds of relationships.
  - “Equivalence”
  - “Another aspect”
  - “Class-instance”

Counterparts - Language Interfaces

- While languages are defined by their verbs, the interfaces are defined by the nouns
  - (why: because interfaces are essentially static – and nouns describe static things).
- Consequently, defining an interface between languages boils down to providing a mechanism to associate nouns in different languages in some way
  - Different aspects of the same underlying thing.

Tools and MVC

- This has a big implication: Modeling tools should adhere to the MVC paradigm. The concept represented by a noun must be independent of the noun itself. A noun is merely a representation of a concept. The concept may be only one aspect of a larger thing.
- While this may seem obvious now, this is something we missed when Honeywell released the original version of DOME. This has been corrected in the **new** version of DOME currently under development.

One final note

- In order to provide appropriate support for this kind of mapping, tools should have some underlying meta-modeling capability.
- In all likelihood, this means complete meta-modeling support.