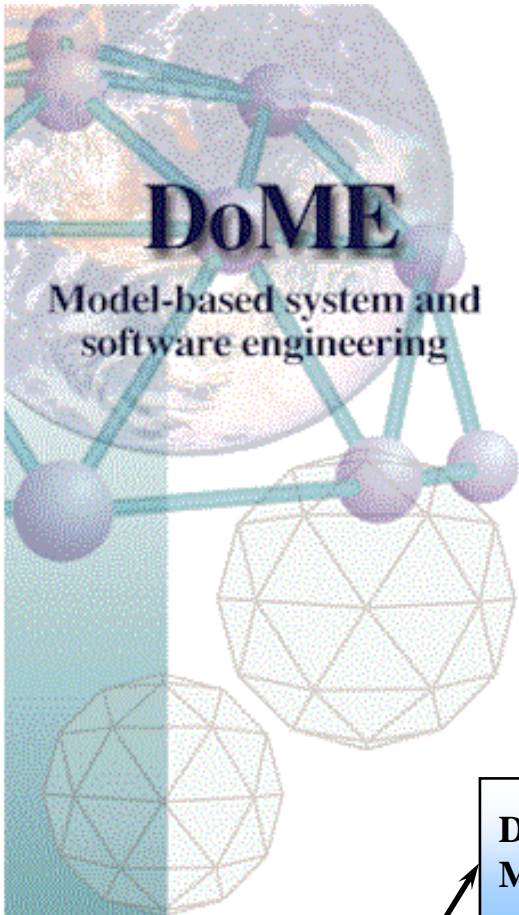


DoME (Domain Modeling Environment) **Application to the AADL**

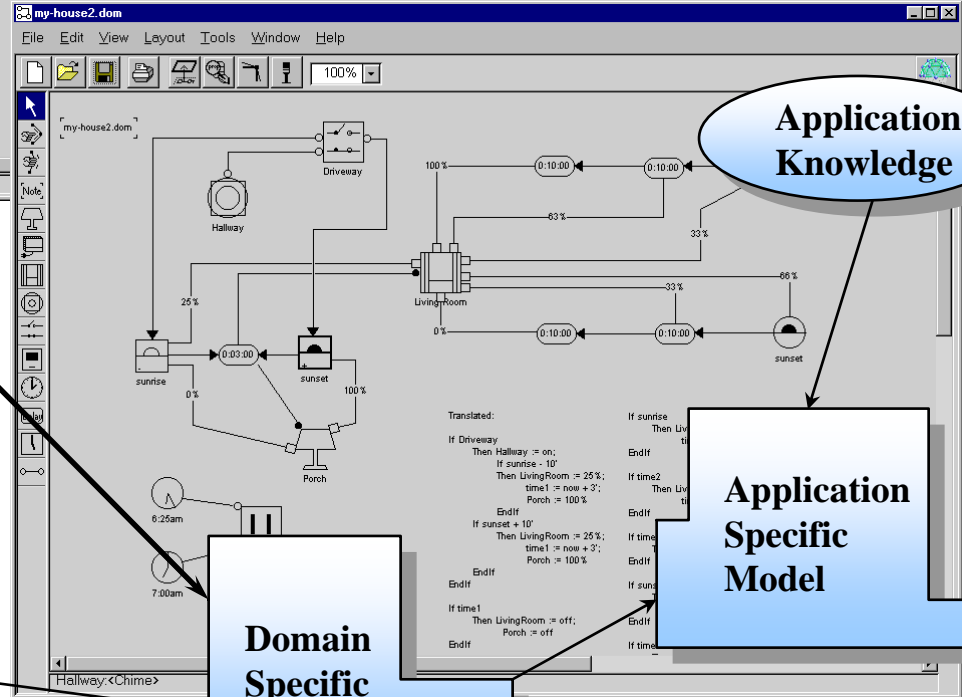
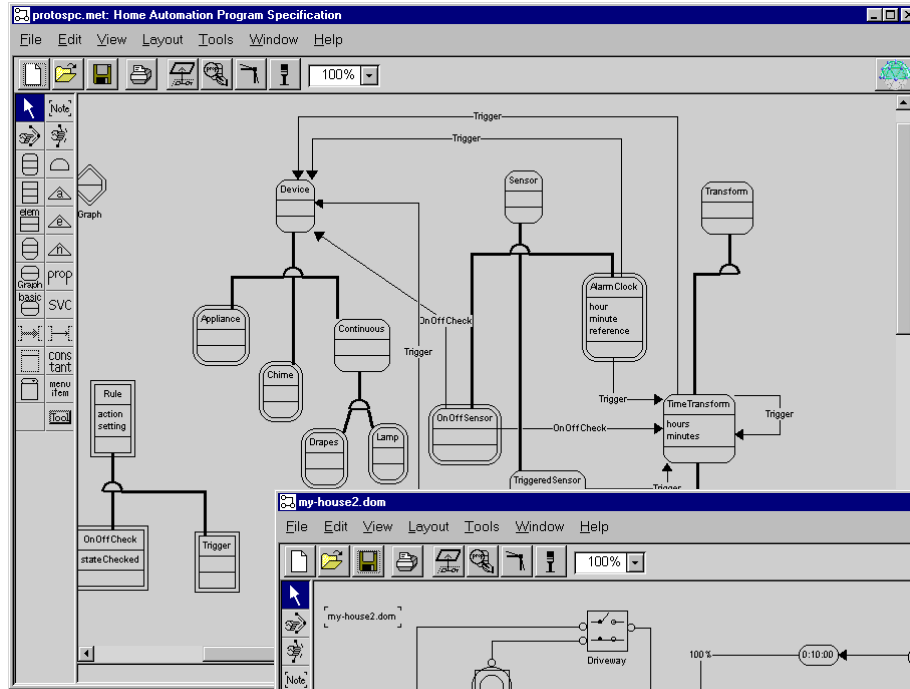
Primary Contact:

Eric Engstrom

eric.engstrom@honeywell.com



DoME
Model-based system and software engineering



Domain Knowledge

Domain Specific Meta-Model

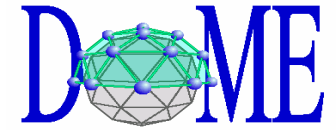
Product Family Knowledge

Domain Specific Model

Application Knowledge

Application Specific Model

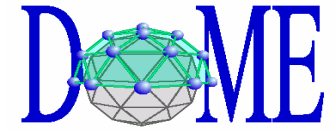
What Is It?



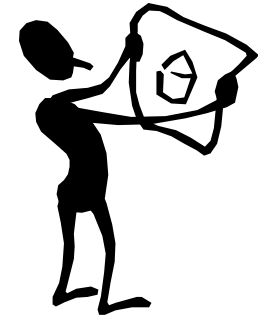
- **A Generic Model Processor**
 - Meta-model based
 - Supports complex graphical syntax
 - Dozens of notations
 - Standard and user-defined properties on all elements
 - Component shelf (library of patterns for reuse)
 - Handles very large models (> 10,000 objects)

- **Two Implementations:**
 - Smalltalk – under development since 1990
 - Java – rearchitecture / reimplementaion started in 2003

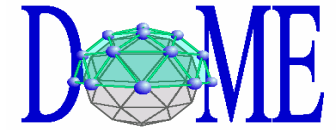
DOME Builds Tools That Build Systems



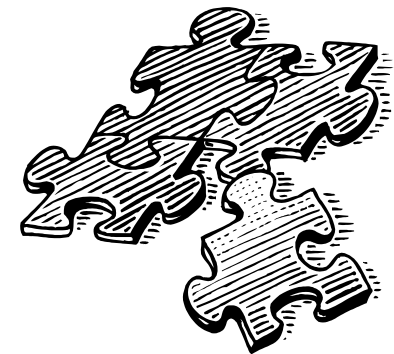
- **DOME's modeling capabilities enable/support:**
 - **Capturing the domain-knowledge of systems:**
 - requirements capture and analysis
 - design patterns and design expertise
 - coding, parameterization, table setup
 - test case generation, testing, test management
 - **Specification of systems**
 - applying the built-in tools, or
 - building domain-specific tools (quickly and easily)
 - **Execution/Simulation**
 - because of the domain knowledge built into the tools
 - **Integration with COTS and other tools**



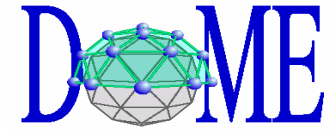
Key Features



- **Meta-Modeling to capture domain knowledge**
 - UML-like (Smalltalk version)
 - MOF-based (Java version)
- **Concrete / Graphical Syntax specifications**
 - “By-example” graphical syntax creation (Java)
- **Model interpreter**
 - ALL models evaluated on the fly
 - Evolution of (meta-)models can be done at the same time
- **Extension / Plug-in Language**
 - Simulation/Execution
 - Extension of the base functionality
- **Template Language**

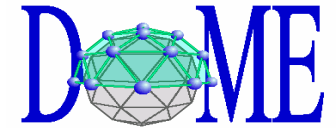


History of DOME

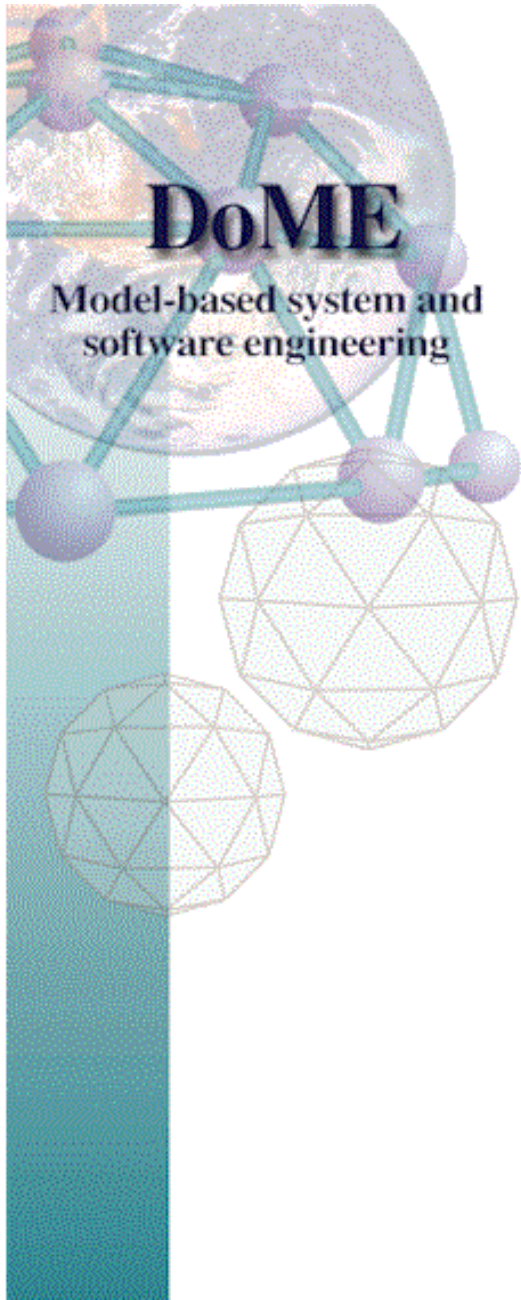


- **15 year history**
 - DARPA, NASA, internal, open-source collaborations
- **Evolutionary experimental platform**
 - Many experiments, each with its own direction
 - Many projects, each with its own requirements
- **True “DOME” foundations date back to 1992**
 - ProtoTech (aka CPL) → DOME
 - DSSA → MetaH

DOME Evolution



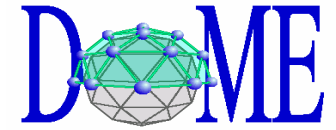
- ...key to understanding DOME's current architecture
- First "DOME" (as yet unnamed) started as a Petri-Net tool
- Build a variety of other modeling tools before meta-ifying the generation of tools
 - MetaDOME
- moved from generation to interpretation
 - ProtoDOME
 - CyberDOME
- currently moving to a completely new architecture and implementation in JAVA
 - Interpreted meta-models, ala ProtoDOME



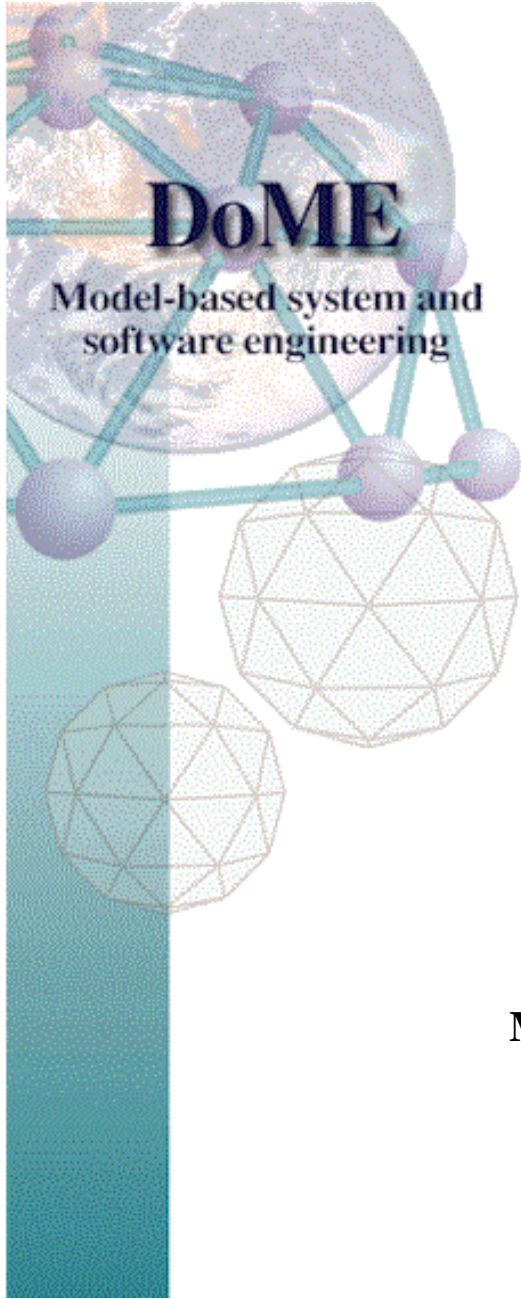
DoME Example #1

Petri-Net
Using Smalltalk Version

MetaH / DOME interactions



- **Graphical-MetaH was an experiment during ProtoTech/DSSA contracts**
- **Interesting features included**
 - complete w.r.t. textual MetaH
 - generation of textual MetaH
 - integration with MetaH toolset
 - Compiler link w/ traceability of errors back to specifications
 - Archetypes and shelf concepts developed
- **NO explicit separation of declaration from instantiation**
 - “Trace” or “binding” through declaration space for instantiation
- **Lacking**
 - **Textual MetaH to Graphical connection**
 - [actually, had a parsing capability, but it was rudimentary]



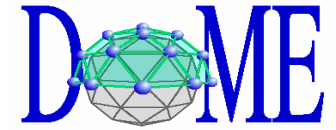
DoME Example #2

Old Graphical Meta-H Toolset

MetaDoME-generated on top of Smalltalk foundations

Not Interpreted!

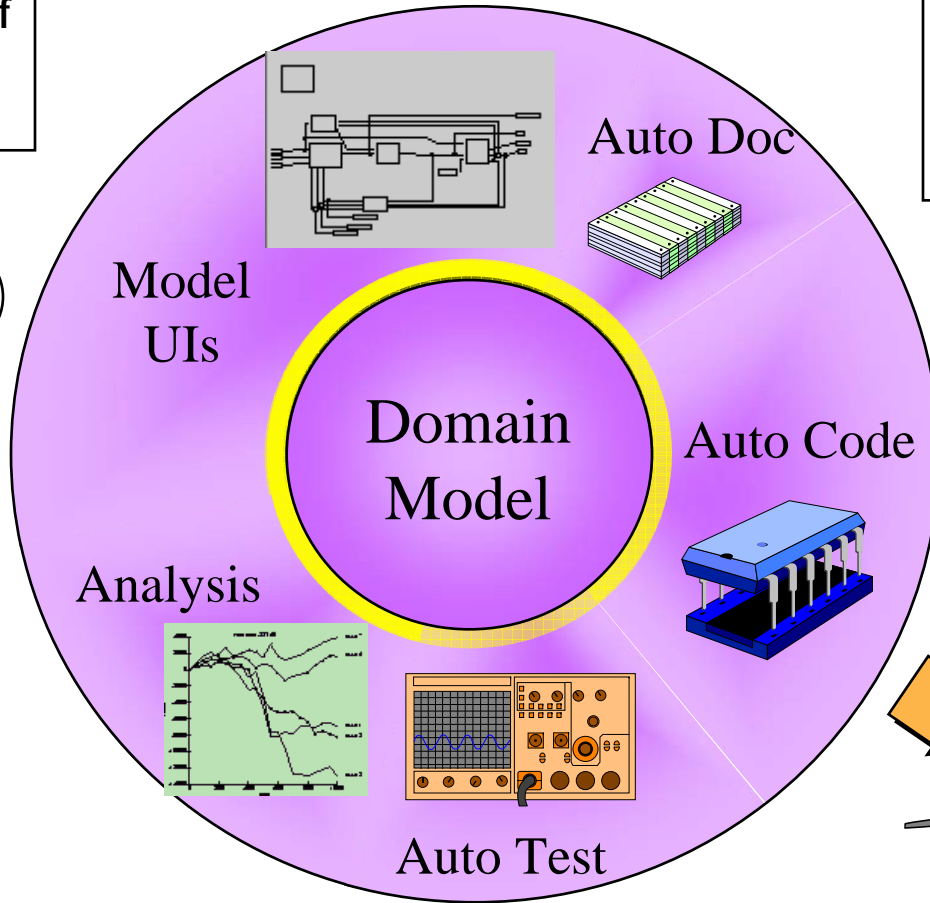
Antiquated Model-Based Development View



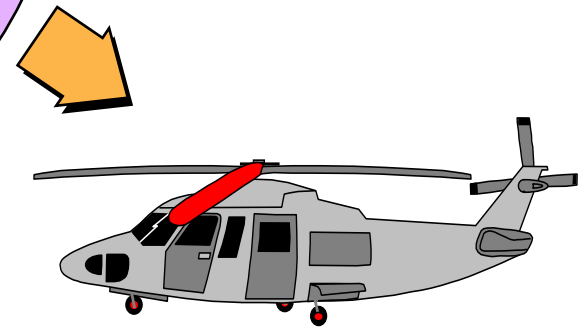
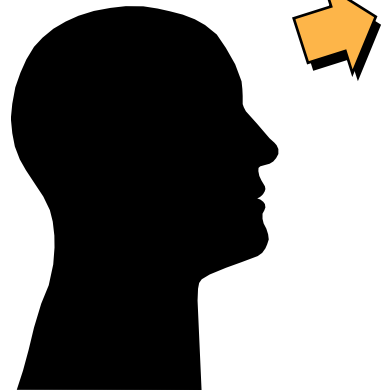
Rapid development and customization of domain-specific modeling tools!

Powerful scripting language for implementing constraints and semantics!

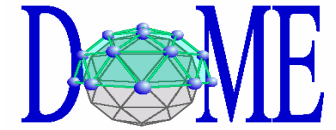
Requirements, architecture, design, test strategies



COTS-tool-based artifact generation environment!

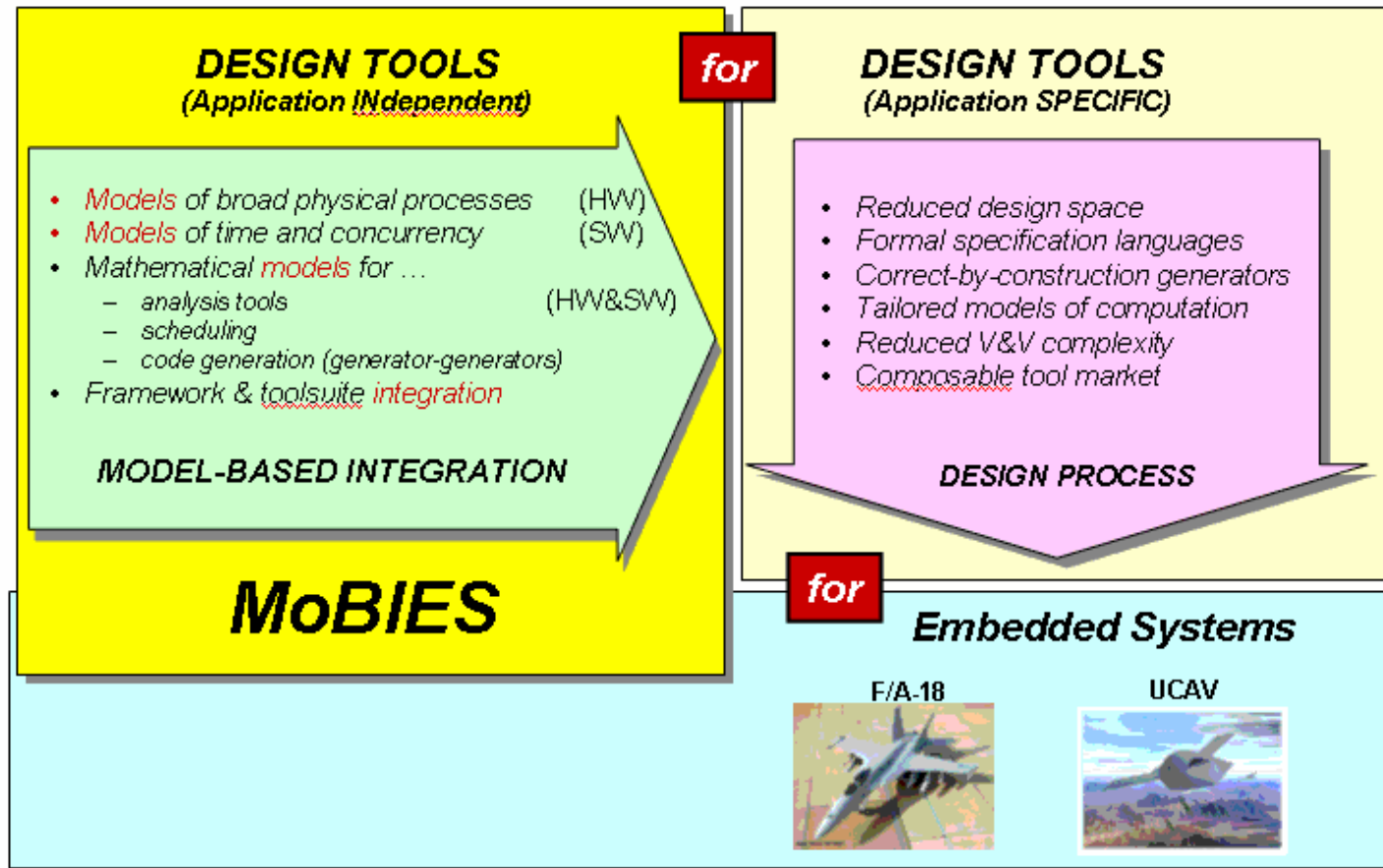
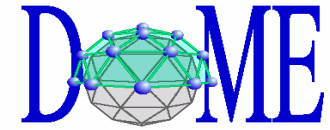


... that was 10 years ago



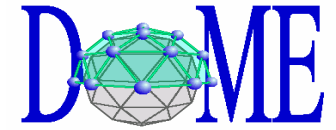
- **In the meantime,**
 - many other meta-modeling tools have been developed
 - GME (Vanderbilt), MetaEdit (MetaCase Consulting)
- **Everyone has meta-models**
 - MOF, XMI, EMF, etc...
 - Even XML has schemas, which are weak meta-models
 - Everyone interprets some form of meta-data
- **At Honeywell, a lot of work has gone into using meta-modeling to develop domain-specific languages**
- **Recently, we participated in DARPA's MoBIES program**

DARPA MoBIES Program



“MoBIES is developing interoperable tools to design and test complex computer-based systems such as avionics, weapons, and communications systems. These tools will simplify the design of complex embedded systems by focusing on the pre-production environment rather than after-the-fact integration. The approach is to customize the design tools used by applications engineers so that controller design and systems integration can be more fully automated and the errors thereby reduced.”

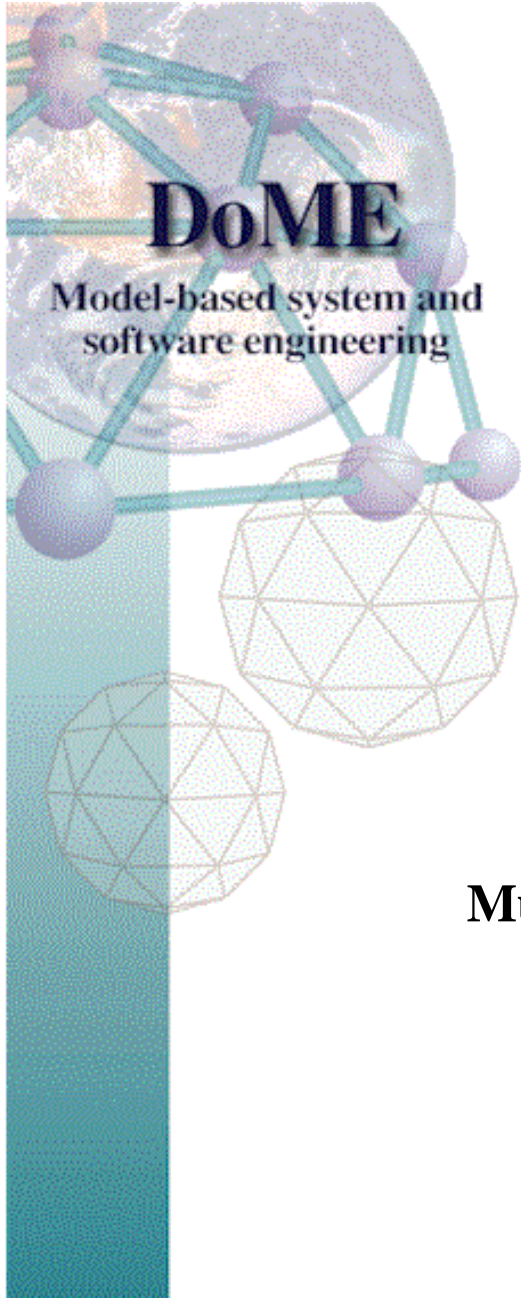
MoBIES Results



- **Pulled and sculpted DOME about as far as it could go**
 - Multi-view support
 - Composable-code generation
 - View-Query language
 - COTS tool integration (ROSE and Simulink)
 - Many good “fielded” tools and lots more experiments
- **Smalltalk foundation showing age:**
 - Systemic architectural issues
 - Tied meta-model and visualizations
 - Lack of developers
 - Standards evolved beyond DOME
 - Not UML
 - Comparative lack of library development

- **Developers relatively easy to find**
- **Based upon current standards**
 - ... therefore lots of libraries to build upon
- **Benefit of hindsight to correct past failings**

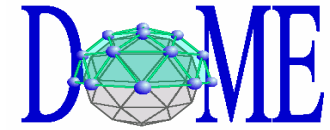
- **Able to consider required features to avoid “center of the universe” hole:**
 - Tool-chain agnostic / COTS tool integration
 - True view/model independence
 - Multi-view / view-query fundamental to architecture
 - Meta-model agnostic (if you want to code your own “engine”)



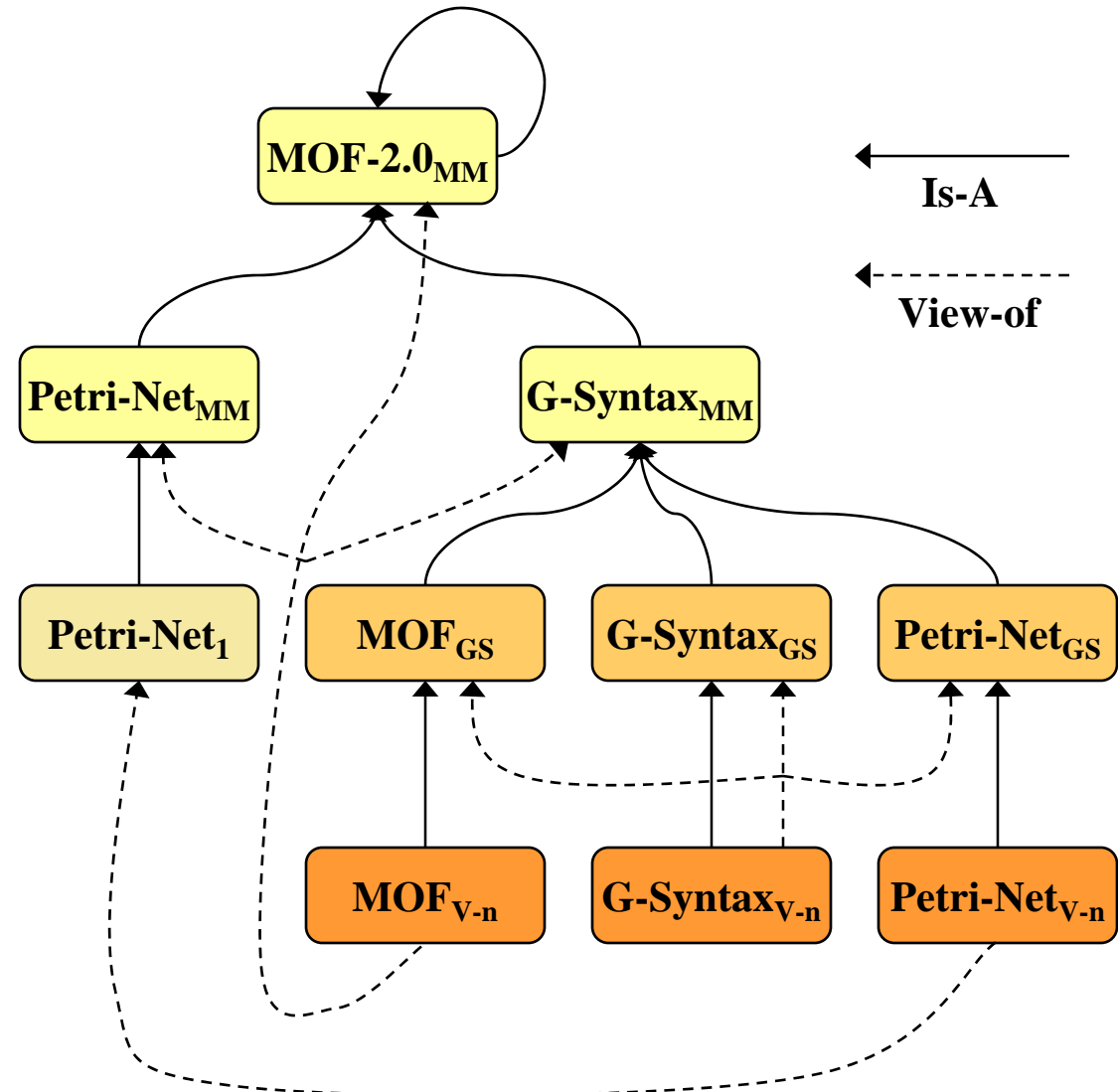
DoME Example #3

Multi-view modeling and COTS tool integration

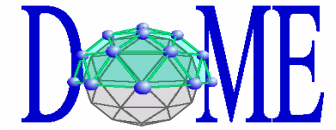
DOME-Model Hierarchy



- **MOF-Based**
- **Meta-Circular**
- **Everything is a model**
 - Views are models
 - Syntax-Specifications are models too
- **“DOME Data Model”**
 - Model-agnostic
 - “Engines” used for interpretations

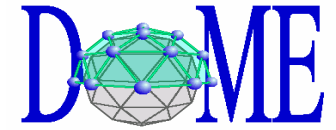


DOME for AADL approach



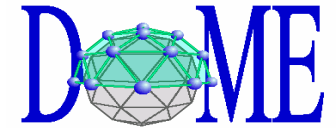
- **Integration with OSATE/Eclipse**
 - **EMF is the key element here**
 - “Live” integration with in-core representation gives biggest benefit
 - Avoids disconnect/generation/parsing issues
- **Already have a standalone Swing-based GUI**
 - **Add DOME as a plug-in to Eclipse**
 - **Possibly use SWT rather than Swing/AWT**
- **Must maintain independence from EMF specifically**
 - **Have other integration approaches in the works**
 - **Maintain DOME-”native” representations too**

EMF key element / potential barrier

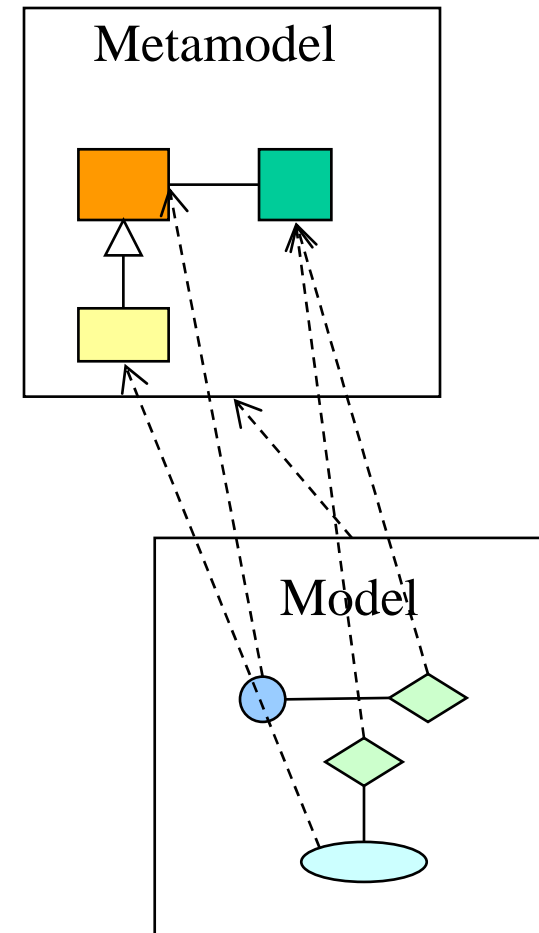


- **EMF meta-models are (largely) equivalent to DOME meta-models**
- **Trick (from our perspective) is to interpret these directly**
 - don't want to tie ourselves to a single language (AADL)
 - want to allow users to use the AADL annex capabilities and still be able to use DOME to graphical render these models
 - DOME graphical syntax specifications remain independent of underlying in-core representation
 - may be able to extend/subclass our underlying data-model representation (which is independent of the meta-model being captured) to dynamically wrap EMF concepts given an associated EMF meta-model.
- **Developing rough prototypes of key elements to help scope work required.**

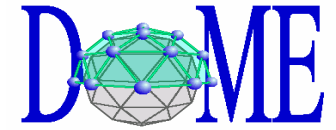
DOME Metamodel Paradigm



- **Metamodels describe all models that can be created**
 - Define the domain
- **Every model is an instance of a metamodel, every model object is an instance of a metatype (an object on the metamodel)**
 - Similar to objects as instances of classes

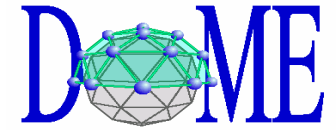


DOME Metamodel Paradigm II



- **Metamodels specify:**
 - Types of objects
 - Types of connections between objects
 - Properties (name, type, parameters, etc.)
 - Constraints (what is and is-not legal)
- **Metamodels are dynamically interpreted**
 - Changes to a metamodel are immediately reflected in instance models

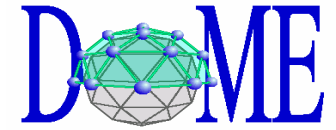
Payoffs



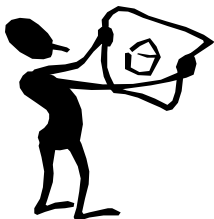
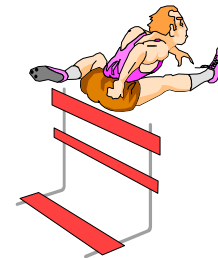
- **Methodologists can quickly formulate domain models and easily evolve them.**
- **Tooling is less risky to develop and can evolve with the domain models.**
 - **Powerful notations (views) can be developed for a model in minutes to days.**
 - **Analysis functions and transforms increase the model's value**



Value and Dividends



- **DOME significantly lowers the cost of making and keeping tweekable, model-based tools for specialists, leading to ...**
- **More tools that are right for the job**
 - Faster cycle times, higher quality, better maintenance
- **Enabling System-Level Integration**
 - More precise and complete system specifications
 - Integration with COTS tools
- **Complete with artifact generation capabilities**



Formalized, Model-Based Development Paradigm

