Model-driven, Context-aware Semantic Governance of Dynamic Systems

DANMS 2012 Keynote

Joel J Fleck II
Senior Standards Architect, HP Office of Standards, Global Technology Program, HP Labs
joel.fleck@hp.com

Dr. John Strassner
Chief Technical Officer, Software Labs, Central Software Institute, Huawei Technologies
john.sc.strassner@huawei.com
Outline

• **Challenges**

• Top-Down Business-aligned Model Driven Generation

• Bottom-Up Model Driven Generation

• Conclusions
Joel’s Rules of Next Generation Management

1. No matter how dynamic and adaptive your network and services are designed, they are basically static if your management is static.

2. Management designed independently from what it is managing is useless.

3. Management must reflect the goals, rules, semantics and configuration (i.e., the models) of what it is managing.

4. Managing things outside your Scope of Authority is futile!!

5. Management components must be designed so that each is dynamically adaptive to the current problem and context.

6. Dynamic management without context is blind!!

7. Management that does not reflect the Business of the System (from all roles) is management for the sake of management.
Need to:

- Get Business People actively involved in Modeling Process,
- Move away from static models and “one-time” transformations,
- Provide capabilities for Semantic Reasoning between Models,
- Incorporate Vocabulary (Information/Data), Constraints (Rules/Policies), and Relationships (Capabilities/Operations) for both the Management and Application into the Model-Driven Approach,
- Assure that underpinning technologies (e.g., policy, models, and processes) are dynamic and adaptive,
- Provide contextual information/tools to support analytics, decisions and stimuli monitoring,
- Facilitate all this adaptability.
Solution

• Move from using fixed “top-down”, “bottom-up”, or “design-driven” approaches to an approach which combines all thus facilitating adaptation,

• Provide the modeling and transformation tools and mechanisms which ease the process for all parties.

**NEED:** An end-to-end model-driven approach that facilitates:

- **Generation of models** (ontologies, information, policy, computational, technology neutral and technology specific)
- **Transformation to higher and lower level models,**
- **Traceability of information throughout models,** and
- **Analysis of compatibility/consistency of inter-domain semantics.**
Outline

• Challenges
• **Top-Down Business-aligned Model Driven Generation**
• Bottom-Up Model Driven Generation
• Conclusions
Model Driven Approach

Semantic
- Model Ontologies and Semantics

Business Visualization
- Business Terms and Relationships

Business Language
- Business Vocabulary, Goals, Rules and Context

Arch/Design
- Technology Neutral Models of Policies, Data and Capabilities

Implementation
- Implementations of Policies, Data and Capabilities
Model Driven Approach

Business Visualization

Business Terms and Relationships
A Concept Map is a visual representation of a Business Model in the form of propositions.

Propositions are in the form of a triple:

Concept ------ Relationship ------ Concept

Where each Concept is:

noun or proper name

And Relationship is a Verb Expression of the form:

Relationship Expression [Quantification Expression]*

Relationship Expression is of the form:

verb [, verb] [negator] [limiter] [conjunction/disjunction]

and Quantification Expression is of the form:

[quantification] [conjunction/disjunction] [quantification]

* Note: Blue Text denotes extended Verb Expression capabilities from standard Context Mapping tools.
Sample Concept Map
Model Driven Approach

Semantic

Model Ontologies and Semantics

Business Terms and Relationships

Business Visualization
Ontologies? What Are They and Why Do We Need Them?

- Ontologies are a formal representation of the concepts of a domain and the relationships between those concepts.

- Ontologies are derived via transformation from Concept Map Business Visualizations and represented in OWL 2.0.

- Ontologies are used to reason about semantic compatibility within/between domains and assure consistent conceptual and relationship usage between domains. Thus by monitoring the ontologies of inter-operating domains, we can detect potential problems/incompatibilities before they occur.
Model Driven Approach

Semantic

Model Ontologies and Semantics

Business Terms and Relationships

Business Visualization

Business Vocabulary, Goals, Rules, and Context

Business Language
Semantics for Business Vocabulary and Rules (SBVR)

➤ SBVR Vocabulary:
  - Built using these basic types:
    » Term (Object Type): general concept in form of noun
    » Name (Individual Type): corresponds to only one (proper named) object
    » Fact Type: denotes some type of relationship between two or more Terms or Names or defines a characteristic of a Term or Name
    » Role: A Term that corresponds to things based on their playing a part, assuming a function or being used in a particular fashion

➤ SBVR Rules:
  • Based on Fact Types:
    » Alethic rules which define structure of organization and must be followed (necessities, possibilities, or impossibilities)
    » Deontic rules which define operations of organization (must be verified) (obligations, permissions, prohibitions)
Concept Map to SBVR Transformation

- SBVR Vocabulary generation from Concept Map Propositions:
  - Creates SBVR Vocabulary from Concepts and basic Concept Relationships:
    - Terms (general concepts),
    - Names (Proper names, specific individual concepts),
    - Facts (expressions relating 2 or more terms or Names)
  - Corrects syntax for all entries in Vocabulary as required by SBVR editing tool,
  - Eliminates duplicate entries,
  - Negation expression in Proposition creates negated Fact,
  - Inserts appropriate indefinite articles to improve Fact readability,
  - Handles partitive articles/fact limiter keywords (e.g., “some” and “each”),
  - Comma-separated multiple verbs in Proposition creates separate Fact for each verb
Concept Map to SBVR Transformation

Verb Expressions are transformed to *suggested* SBVR Business Rules as follows:

- For each Rule created, the associated SBVR Fact is created (if it does not already exist),
- Verb Expressions with negation:
  - Transformed to either SBVR *impossibilities* (Alethic) or *prohibitions* (Deontic),
- Relationship Expressions with non-negated Fact limiter:
  - Unconditional limiter (“each”) transformed to SBVR *obligations* (Deontic),
  - Conditional limiter (“some”) transformed to SBVR *permissions* (Deontic),
- Non-negated Quantification Expressions:
  - Definite Quantifications transformed to SBVR *necessities* (Alethic) or *obligations* (Deontic),
  - Indefinite Quantifications transformed to SBVR *possibilities* (Alethic) or *permissions* (Deontic),
- Simple (non-Compound) Quantification Expressions:
  - Transformed to Structural (Alethic) Rules,
- Compound Quantification Expressions:
  - Conjunctions (“and”) transformed to Structural (Alethic) Rules,
  - Disjunctions (“or”) transformed to Operational (Deontic) Rules.
Management Contextual Information Generated:

- Adjacency Matrices (for both specifically defined connections and specifically denied connections) for path analysis, connectivity analysis, …
- Weighted Connectivity graphics files (for end-to-end connectivity analysis, node impact analysis, node connectivity analysis, node cluster analysis, and process identification.)
Model Driven Approach

Semantic
- Model Ontologies and Semantics

Business Visualization
- Business Terms and Relationships

Business Language
- Business Vocabulary, Goals, Rules and Context

Arch/Design
- Technology Neutral Models of Policies, Data and Capabilities
UML Transformation from SBVR
Model Driven Approach

Semantic

Model Ontologies and Semantics

Business Terms and Relationships

Business Vocabulary, Goals, Rules and Context

Business Visualization

Technology Neutral Models of Policies, Data and Capabilities

Business Language

Implementation

Implementations of Policies, Data and Capabilities

Arch/Design
What About Transformations for Policies?

- Proof-of-concept done by Sven van der Meer (Ericsson - Ireland) based on his SKB ANTLR based tools
- Parses SBVR Rules (using SBVR Vocabulary also) and produces N3 (could also generate Turtle)
- N3 or Turtle are used as a Technology Neutral Representation for general purpose Policy Languages (e.g., KAoS, Ponder, Rei, …)
- Also includes capability to generate Traceability Maps for use in Management Analytics
Outline

• Challenges
• Top-Down Business-aligned Model Driven Generation
• Bottom-Up Model Driven Generation
• Conclusions
But ----

What About the other Part (i.e., Bottom-Up Changes in Configuration, Deployment, and Other Aspects)?

---- John
John’s Rules of Next Generation Management

1. The customer will always test and retest code changes. Code changes represent instability. Data changes are much more acceptable.

2. Context must be used to actively select the working set of applicable policies for a given situation.

3. The ability to dynamically change behavior to adapt to changing user needs, business goals, and environmental conditions will be increasingly important.

4. Management components must be designed so that each is either self-managing and/or knows how to collaboratively help other components manage a less capable component.

5. Management that does not reflect the needs of the Business (from all roles) is useless.
Traditional Approach

Without a model and metadata, when business services change, code changes, and system suffers.

Components do not change, but their semantics do; how are business requirements mapped onto implementation artifacts?

Bridge to rest of System

Business Goals

Business Services

Component Building Blocks

Code, Data, Interfaces

Management and Security

Core Services

Identity, Naming, Registration, Location

Framework Services

Infrastructure Services

Not Frameworks, but Containers, App Servers, and other components

Functionality Hierarchy

Speed of Adaptation

Distribution, Federation, Data Access

Policy, Process, Messaging

Service Composition, Service Orchestration, Service Choreography, Service Automation, Virtual and non-Virtual Resource and Service Management, ...

Management and Security

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Naming, Registration, Location

Component Building Blocks

Code, Data, Interfaces

Core Services

Identity, Nam...
A New Meaning to “Bottom Up”

- Componentized architectures must be able to adapt at different time scales

Model helps capture the changes in semantics required when business services change

Metadata helps capture the changes in semantics required when business services change

Intelligence through Metadata

Middleware

Core Services
- Identity, Naming, Registration, Location
- Core Services
- Management and Security
- Bridge to rest of System

Infrastructure Services
- Policy, Process, Messaging

Framework Services
- Model, Distribution, Federation, Data Access

Component Building Blocks
- Not Frameworks, but Containers, App Servers, and other components

Business Services
- Code, Data, Interfaces
- Business Goals
- Service Composition, Service Orchestration, Service Choreography, Service Automation, Virtual and non-Virtual Resource and Service Management, …
A Meeting of Frameworks…

Top-Down, Business-driven Modeling Framework

Bottom-Up, Model-driven Engineering Framework

Dynamic Behavior Modification *without Code Changes*

Data Abstraction, Mapping, and Persistence Layer

Dynamic Object Model based on DEN-ng
A Dynamic Object Model stores the object model in a repository, and enables the object model to be changed at runtime, thereby changing the behavior of the system using the object model.
Key Model-driven Features

- Dynamic adaptation can be implemented *without recoding, recompiling, and redeploying the system or its modules*
  - Features are not implemented in classes, since when features change, classes change, which causes code to change.
  - Rather, features are described by metadata that are interpreted at runtime; this enables changes to be immediately reflected.

- Resulting systems can
  - support new devices and products in days, not months.
  - add customer requested features faster.
  - be significantly less complex, due to increased reusability and use of software patterns.
  - cost less due to increased productivity, easier development of new features, and increased maintainability.
  - have fewer defects, due to increased use of patterns (code that is repeatedly used and tested, and becomes more robust).
# MDE vs. Traditional Software

<table>
<thead>
<tr>
<th>Feature</th>
<th>Traditional Software</th>
<th>Model-Driven Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer acceptance</td>
<td>Viewed as potentially unstable</td>
<td>Metadata viewed as <em>data, not code</em></td>
</tr>
<tr>
<td>Product version coexistence</td>
<td>Requires special handling</td>
<td>Version info is metadata!</td>
</tr>
<tr>
<td>Changes implemented by</td>
<td>Developers only</td>
<td>Domain experts for changes supported by model</td>
</tr>
<tr>
<td>Changes verified by</td>
<td>Developers, then QA, then approved by end-user</td>
<td>Same, <em>but can support non-developer through tools, so faster</em></td>
</tr>
<tr>
<td>Adding features to an existing supported device</td>
<td>Change code; test; recompile, redeploy; requires weeks+</td>
<td>Use modeling patterns and/or metadata and/or DSLs; requires days</td>
</tr>
<tr>
<td>Adding support for new devices</td>
<td>Change code; test; recompile, redeploy; requires weeks-months</td>
<td>Use modeling patterns and/or metadata and/or DSLs; requires days</td>
</tr>
<tr>
<td>Adding support for app-specific calculated data</td>
<td>Change code; test; recompile, redeploy; requires weeks+</td>
<td>Use modeling patterns and/or metadata and/or DSLs; requires days</td>
</tr>
<tr>
<td>Adding support for third-party devices &amp; software</td>
<td>Change code; test; recompile, redeploy; requires months+</td>
<td>Use modeling patterns and/or metadata and/or DSLs; requires days</td>
</tr>
<tr>
<td>Adding support for software upgrades/patches</td>
<td>Change code; test; recompile, redeploy; requires weeks-months</td>
<td>Use modeling patterns and/or metadata and/or DSLs; requires days</td>
</tr>
</tbody>
</table>
Keys to Avoiding Recompiling (1)

- Metadata: If something is going to vary in a predictable way, store the *description* of the variation so that it is easy to change
  - DEN-ng is the *only* standard model that provides metadata for all entities
  - *Object behaviors can be specified and interpreted at runtime*

Objects that do NOT have a distinct existence, enabling data to be transformed to information and then to knowledge by adding context and semantics to values

Objects that have a separate and distinct existence; not just a collection of attributes or an abstraction of behavior

Information that describes, but does not directly contribute to or impact, the state of the Entity or Value that the MetaData is applied to. CODE IS DATA, DATA IS CODE. DATA CAN DRIVE BEHAVIOR.
Metadata-driven Architecture

- An architecture that dynamically adapts to change
  - DEN-ng metadata model describes the characteristics and behavior of all managed entities (classes, relationships, constraints, behaviors) as objects
  - This enables the object model to be interpreted at runtime
    - When metadata changes, the system immediately reflects those changes without requiring code changes
    - Applies to business rules and new features (e.g., new devices)
    - Objects have states and respond to events by changing state; the object model defines the objects, their states, the events, and the conditions under which an object changes state
    - Object model can be stored either in XML files or in a database
    - Interprets an object-oriented representation of <> at runtime; <> can be stored in a separate repository and loaded on demand
DEN-ng provides a robust policy models that can be used to implement domain-specific languages as well as robust Policy Languages.
Keys to Avoiding Recompiling (3)

- Create a fabric of patterns that enable different types of classes, attributes, relationships, and behavior to be interpreted at run-time
  - The fabric enables features to be implemented as reusable templates, so different applications can solve their problems in similar ways

![Diagram showing various patterns and relationships]
Dynamic Object Model Implementation

- Four ways to implement
  - Stitching together instances
  - Using metadata to define instances
  - Using an interpreter
  - A combination of the above

- Each of these methods can support rules from any part of the policy continuum; metadata is easiest and most powerful

- Advantages
  - Two primary ways to change behavior: (1) through metadata and (2) by changing the interpreter of the metadata
  - Patterns can be used for each of the above 4 implementations
  - Metadata enables informal and formal data to be used to customize behavior dynamically at runtime
  - Rules can be used to define entities, attributes, relationships, constraints, entity validation, choosing between parameters or algorithms…
  - This can be generalized into event-driven systems, workflows…

- Disadvantages
  - Programmers not used to any of the above methods
Dynamic Model Persistence

- Changes can be temporary or permanent
  - Metadata are regular objects, and can be stored as such

Data Abstraction Layer (DAL)

DAL Persistence Mechanisms

RDBMS  Directory  NoSQL  File System  Metadata

System being Controlled

Metadata Interpreter

XML Parser
Model and DAL Differences

Mapping Tier
Provides unified access to heterogeneous data sources

Transformation Tier
Data- and service-based mapping, metadata-based query rewriting, support for data access patterns

Distributed Object Access Tier
Distributed queries, data pre- and post-processing, metadata-based reflection (observe and modify its structure at runtime)

Repository Adapter Tier
Support for queryable data stores using native and generic drivers (future: support non-queryable and language-based)

Presentation Tier

Storage Tier

DEN-ng

RDBMS
NoSQL
Data Access Layer Translation

Presentation Tier: Get the number of alarms for one week

Data Access Layer:

getData(alarmStatus, status > 2, today, today-7)

**to SQL**

SELECT COUNT(*)
FROM AlarmsTable
WHERE Alarms.Status > 2 AND SYSDATE > SYSDATE - 7

**to MongoDB**

db.alarms.find ( {
  ( ( status: {$gt: 2} ).count( ) ),
  ( $gte: new Date DATENOW,
  $lte : new Date (DATENOW – 7) )
})

Further optional SQL optimization

Further optional NoSQL optimization

Storage Tier
A Future Management System (1)

- Open Source Technologies
- Standards-based Technologies
- Metadata-driven Behavior
- “Smart Objects”
- Model-driven

Model-driven, Content-aware, Semantic Governance of Dynamic Systems
A Future Management System (2)

System Being Managed

- Client Tier
- API Tier
- Services Tier
- Management Tier
- Model Tier
- Storage Tier
Linking the Two Frameworks (1)

Top-Down, Business-driven Modeling Framework

SBVR

Expresses business vocabularies, rules, and semantics in natural language

Bottom-Up, Model-driven Engineering Framework

Model-driven Architecture

Map computation-independent and platform-independent entities to technology-neutral architecture

Models, Ontologies, FOL, and Metadata

Model-driven Transformations

Map to various standards-based vendor-independent technologies

Model-driven Transformations

Map to various standards-based vendor-dependent technologies

NOT TRADITIONAL MDA!
Implications

- Can coordinate UI(s) with behavior and data, since metadata can specify both

  - SBVR → Table structure with integrity constraints
  - SBVR → GUI control and layout spec and validation

- Can define new entities (e.g., Roles) and instances (e.g., Customers and Services); can define behavior for both

  - Feedback to SBVR authors
  - Entity discovery and validation
  - Create entities, attributes, relationships, instances,…
  - Dynamically add behavior to manage new entities (e.g., using state machines and workflows)

  - Feedback from SBVR authors to developers
Data Integration/Normalization

Other Models → Model Mapping / Transformation → DEN-ng → DEN-ng DDL/DML API

Transformation and Validation → MOF-XSD Transformation

Schema Mapping Rules → Schema Generator

Native Data Loader Utility

SQL Schema Optimization Rules

PostgreSQL Schema

MongoDB Collections

NoSQL Optimization Rules
Novel Semantic Reasoning

1. No Relationship defined between Customer and Device fault or alarm!

2. Infer a Relationship between fault for Customer and Device

3. Strengthen the proof of a Relationship between fault for Customer and Device

Different Languages for Monitoring vs. Configuring

interface FastEthernet0/1
no shutdown

1.1.1.1.1.1.1=34000
1.2.1.5.1.1.10=true
1.3.3.1.1.2=230000

Knowledge Representation and Reasoning
Advantages

- Can address all constituencies of the business
  » Each can use their own concepts and terminology; results can be harmonized
- Enables *domain experts* to assume responsibility
- Can evolve into DSLs, structured interpreters, or moderately sophisticated rule-based languages
- Changes can be made very quickly
- *Behavior can be modified at runtime without changing code, taking the database(s) down, or redeploying software*

Disadvantages

- New development and programming paradigm
  » classes do not necessarily represent business abstractions; metadata does
- More complex implementation
  » the interpreter and the model that is being interpreted co-exist
- New development and support tools may have to be built
Questions?

“Create like a god. Command like a king. Work like a slave”
- Constantin Brancusi