Roadmap for Semantics in

Netcentric Enterprise Architecture

Prepared for the Office of the CTO, US General Services Administration

February 2, 2006

Erick Von Schweber
Synsyta LLC
erick@synsyta.com

Executive Summary

A multi-stage roadmap is presented by which to incrementally transform today's static, rigid Enterprise Architecture into a dynamically fluid and fully netcentric architecture that enables automated interoperability without requiring uniformity. A concrete worked example demonstrates the recommendations of the first stage.

Contents

Erick Von Schweber Synsyta LLC erick@synsyta.com

Introduction

Our Approach

What the Roadmap is

The roadmap lays out the incremental development and evolution of a framework for supporting language and knowledge evolution and interoperability. It is the development of this framework for which the roadmap provides prescriptions, proscriptions, advisories and cautions.

What the Roadmap is not

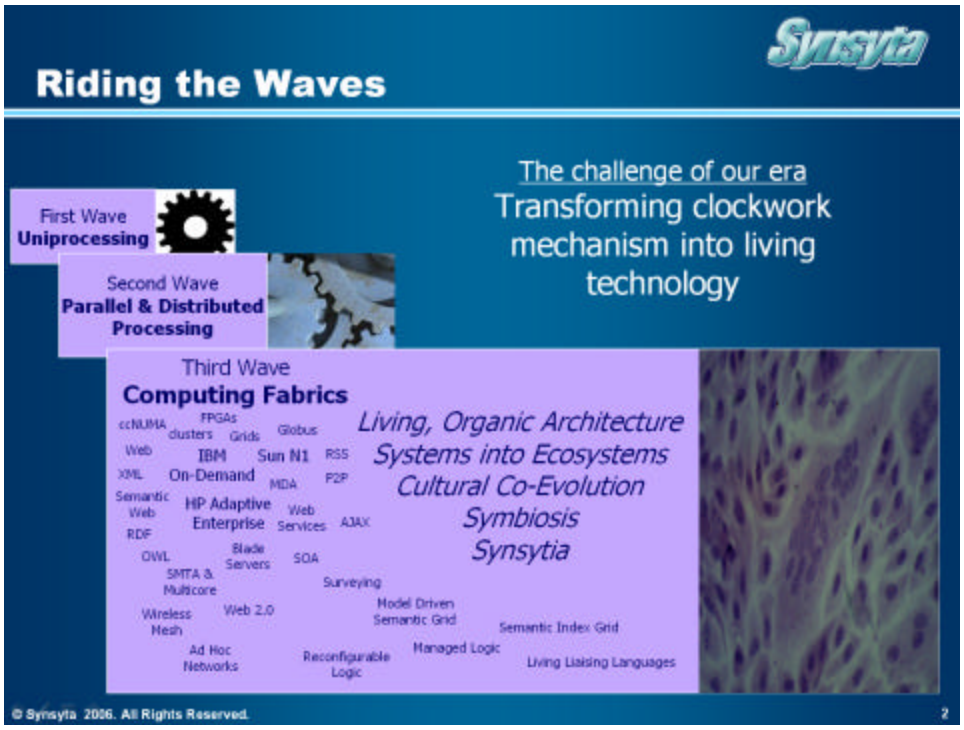
The roadmap is not a rigid prescription - as we drive we may change direction and replan the course ahead.

Motivation - Living, Liaising Languages

Information technology is currently in the midst of a transformation. In 1998 the architects of this roadmap went public with a model of where they saw the industry heading, based on their research and long-time experience in the industry. The model identified a new transformation that was at the time just beginning – the third wave – a transformation into an era we call Computing Fabrics [Von Schweber 1998c]. Whereas the industry began with uniprocessing (the first wave) in the 1940's, and began a transformation into parallel and distributed processing in the late 1970's and early 1980's (the second wave), the third wave, Computing Fabrics, would transform rigid and static distribution of functionality into a fluid, dynamic fabric that blurred the very distinction between system and network, where system boundaries exhibit plasticity.

Nine years later evidence of the third wave surrounds us, in blade servers, wireless mesh networks, Web 2.0, P2P, grid computing, social networks, service oriented architecture (SOA), systems of systems, ..., the list goes on and on.

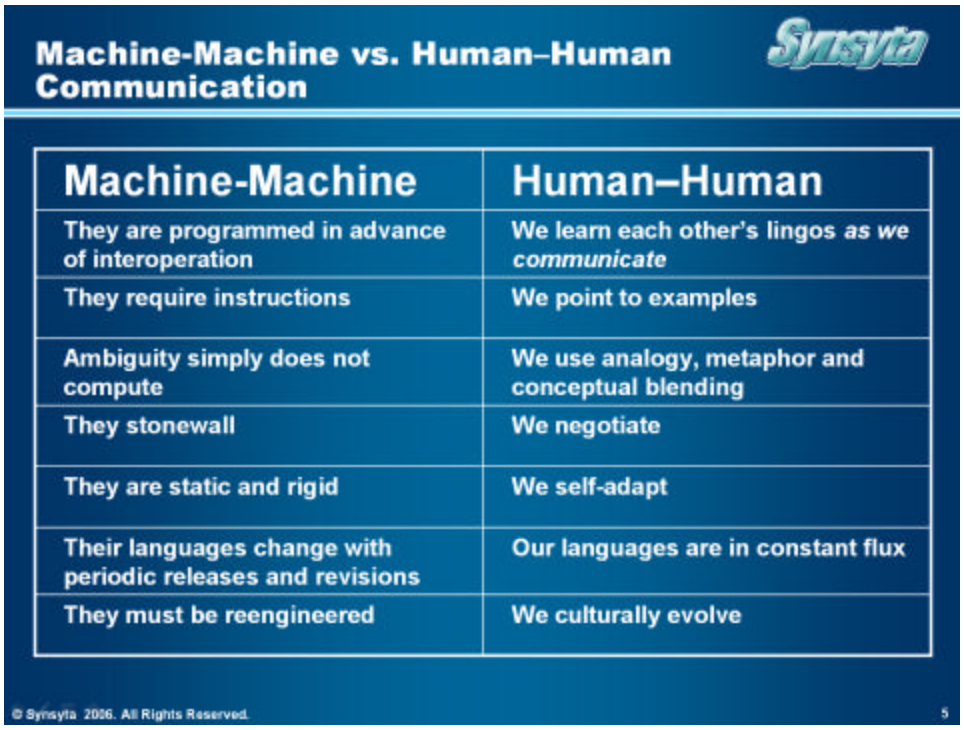
Erick Von Schweber Synsyta LLC erick@synsyta.com



What is not commonly recognized is that this wave is a transformation of clockwork mechanism into living technology. Consider netcentricity; this is not merely the notion of employing multiple services and systems on and across the network. Rather, systems and services will come and go, entering and departing the network, even changing out from under it. Yet the systems of systems comprised of such network resident services must remain functional, continuing to meet their quality of service contracts in terms of capacity and capability. While achieving this is just the beginning it already catapults us into the realm of biological behavior, including metabolism, locomotion and reproduction.

Beyond this, the overwhelming value of netcentricity, collaboration, systems of systems, semantic interoperability and service oriented architectures will only be realized when we acknowledge that systems, to collaborate, must possess plasticity of form and function – they must have the capacity to change, adapt, learn. To get serious value from netcentrism requires more than a mapping of one rigid system to another; it requires self-transformation whereby each constituent may grow in a manner that they may comprehend the other. Each entity achieves this by taking on, as its own, aspects of the other, aligning and merging these new aspects into its being. Each participant must change, learn and evolve.

For me to benefit from your knowledge I must learn, which inextricably changes me, the learner, as to learn is to change and grow. One who is rigid cannot learn and therefore cannot benefit from the knowledge and skills of others. As it is with such social networks so it is with netcentric technologies.



Advanced interoperability requires reciprocal learning whereby each participant becomes more than they were before interoperating; interoperability demands of its participants that they exhibit plasticity, capable of change at an arbitrarily deep level. This is deeply biological. When an ecosystem is extended through introduction of a foreign species new and existing species alike may co-evolve to exploit the fitness landscape of the newly extended ecosystem. Co-evolution is a process of reciprocal evolutionary adaptation through self modification.

Such co-evolutionary symbiosis separates ecosystem from mere system; ecosystem constituents must be able to change, adapt, learn. A system becomes an ecosystem only when its constituents acquire this ability to symbiotically co-evolve. Living, liaising languages are a means to bring this cultural co-evolution and symbiosis to our netcentric technology, including SOAs, systems-of-systems, semantic interoperability, collaboration, etc.

The state of the art in ontologies and ontology languages today are a far cry from living languages. Consider a conversation between two software agents, one based on a reasoning engine using modal logic (Modal agent) and another agent using OWL-DL.

Modal agent: There MayExist x such that [(in(x,y) AND heaven(y)) OR (in(x,z) AND Earth(z))] AND [NOT(in(x,r) AND philosophy(r,OWL)]

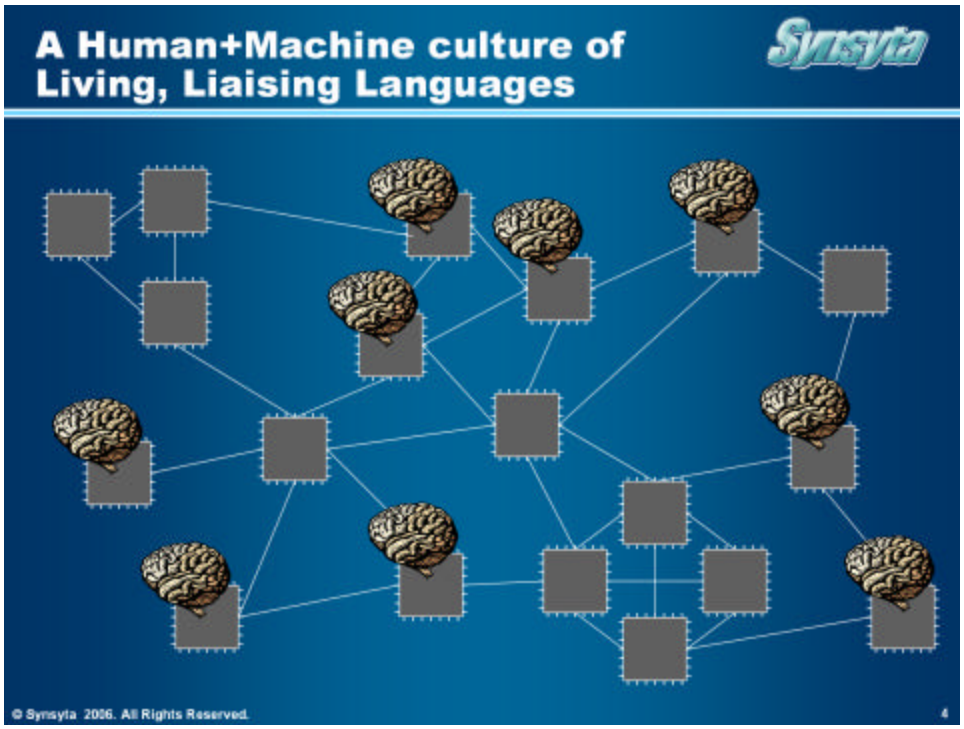
OWL agent: (silence)

(apologies to Shakespeare)

While this fictional exchange was meant to be tongue-in-cheek it is indicative of the kind of netcentric communications we can expect given the best of ontology mediated communication. This is the antithesis of human communication, where the parties will learn and adapt in order to establish a meaningful exchange.

To get the kind of netcentricity worth wanting we must move beyond the clockwork mechanism.

Enterprise Architecture in a netcentric world must acknowledge the era it is in and address the shortcomings and limitations of previous eras' technologies. That means transforming static and rigid languages, ontologies, models, schemas and mappings into their fluid, dynamic, living descendants. We refer to this new breed as Living Liaising Languages.



The question becomes: how are programs, such as GSA's OsEra (Open Source Egov Reference Architecture) to transform themselves incrementally from where they are today to where they need to go?

The roadmap you are reading is a considered answer to that question.

Just the first stage of the roadmap enables automating the mapping of semantics from one community, expressed in their language, to a distinct semantics of another community, expressed in a distinct language. Furthermore, such mappings and transformations may be managed and executed in the mainstream

technology of the present, a relational database management system; no expensive, novel infrastructure required.

Those wishing to bypass the theory and planning and just get a look at a concrete example may wish to jump ahead in this document to the worked example. There we:

- Transform UML class models into OWL-DL ontologies for management in a semantic web repository, to post on the web, visualize with an ontology editing tool (e.g., Protégé), as content for a web page, semantic markup of a web service, or for input to a reasoning engine.
- Transform OWL-DL ontologies into UML class models for management in a MOF repository, visualization and editing in a UML tool, or to apply within a model driven architecture effort.

Erick Von Schweber Synsyta LLC erick@synsyta.com

Prerequisites - what you should know to read this

While this roadmap introduces several advanced mathematical tools, prior knowledge of these, or discrete mathematics in general, is not required. A worked example conveys the power of the tools in the domain of interoperability and Enterprise Architecture. Descriptions of each stage of the roadmap include a summary table requiring absolutely no mathematical skill to understand and appreciate. Each summary table describes the what, why and expected results of each stage, and it does this for three key topics (see below). However, those with a background in computer science or mathematics will be able to get more out of the detail sections of roadmap stages.

How to read and use the roadmap

Think of a road. First you notice that it has multiple lanes, some for faster traffic, some for slower, and one for entering and exiting. Then as you drive the road you notice its on-ramps, which bring new traffic onto the road, and off-ramps, which provide a choice of destinations and an opportunity to rest and refuel. Much later you see new road being built and learn that a road has many layers, where lower layers provide a stable foundation for higher layers and the road surface.

Importantly, as you move ahead on the roadmap, from early to later stages, the character of the recommendations changes. In Stage 1 the roadmap makes certain prescriptions, i.e., "Do this!". But later stages of the roadmap should be taken as suggestions or explorations, not hard and fast prescriptions. The aim of course is to research, verify and flesh-out subsequent stages of the roadmap as we move ahead; we're building the road as we go but we're planning and staging our road crews well in advance of driving.

Structure of the Roadmap

Each stage of the roadmap is accompanied by a summary table. The three columns – Definition, Derivation and Execution – present the key aspects of the framework at that stage.

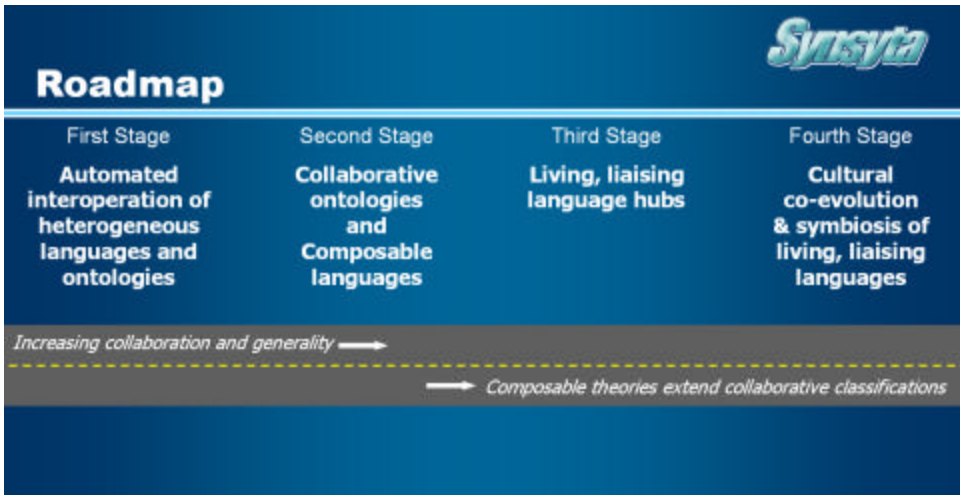
- Definition – Means by which the framework supports defining, revising and extending language systems, including things like the language syntax and semantics, idioms, ontologies, models and schemas.
- Derivation – Means by which the framework supports deriving mappings and transformations between language systems (think translation).
- Execution – Means by which definitions and derivations may be implemented, managed and executed.

Erick Von Schweber Synsyta LLC erick@synsyta.com

This is what each summary table looks like (sans the actual content).

	Definition	Derivation	Execution
What: Prescriptions & Proscriptions; Advisories & Cautions			
Why: Motivation & objectives			
Key tasks			
Expected Results: Improvement above and beyond the previous stage (e.g., comparative advantages)			

The Roadmap at a Glance



Trends over the course of the roadmap

- Evolution of community models from totalitarian dictatorship (one dictated world view at point of departure) through hub and spoke to egalitarian society (each to their own); from homogeneity to heterogeneity; from maximal required commonality to minimal required commonality
- From non-collaborative to collaborative
- From static and rigid to living and evolutionary
- Evolution of meta language from RDF to OWL-DL to OWL-RA to single-math Institution to poly-math Institution to poly-math Parchments
- Evolution of separation of concerns, from syntax to multiple dimensions
- Evolution of infrastructure focus from classifications to theory compositions

Assumptions and preferences

- Exploit available tooling and standards where prudent
- Exploit mindshare
- Build on success
- Evolve incrementally and improve continuously

Erick Von Schweber Synsyta LLC erick@synsyta.com

The Roadmap in Detail

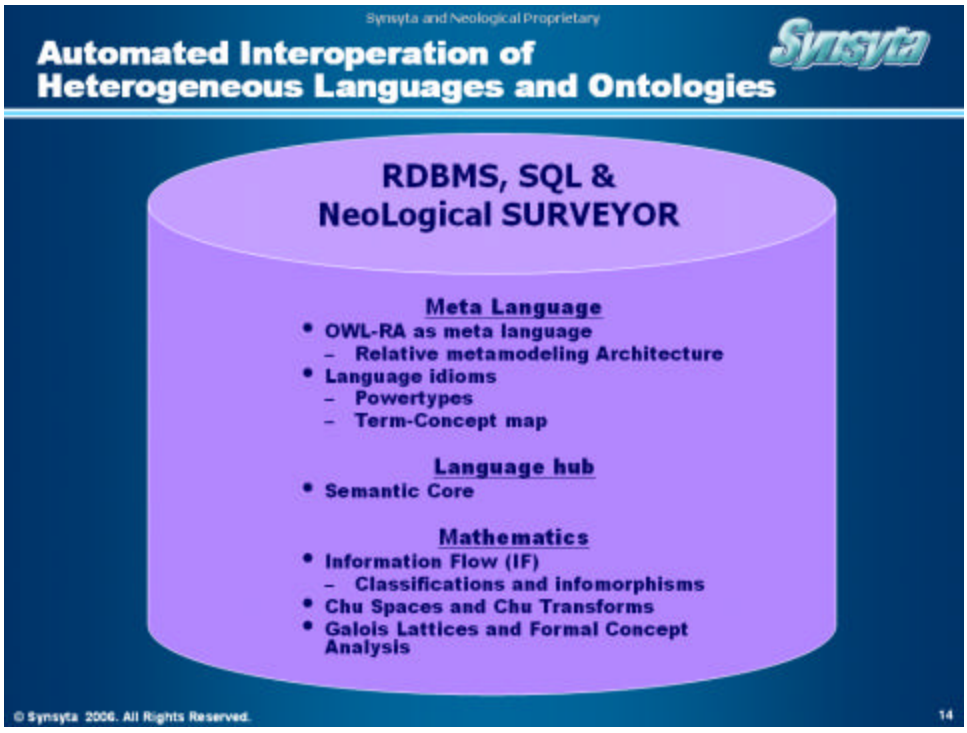
Point of Departure

- Semantic Core [Casanave 2006], i.e., non-composable language, the result of manual analysis and composition, serving as a hub

First Stage – Automated Interoperation Of Heterogeneous Languages And Ontologies

First Stage			
	Definition	**Derivation**	**Execution**
What	Employ an ontology language to capture definitions of community languages *as ontologies*. Capture domain semantics, models, schema and *instance data* within each community's native language(s).	Apply the mathematics of Information Flow (IF), Chu Spaces and Transforms and Galois lattices toward automated derivation of mappings between multiple languages, between multiple ontologies, between ontologies and schemas, and between multiple schemas.	Manage language definitions, domain semantics, and mappings & transformations of these within mainstream relational database management systems. Process and manipulate the above using standard database methods, e.g., SQL.
Why	Tolerate heterogeneity within and across communities while simultaneously managing and manipulating community artifacts in a unified fashion.	Test data, examples and common instance data can provide the reference points from which advanced mathematics can automate the derivation of semantic mappings and transformations.	Exploit commodity infrastructure. Process and manipulate data where it already lives. Build semantic applications following current best practices, skills and tooling.

First Stage			
	Definition	**Derivation**	**Execution**
Key tasks	Extend OWL-FA into OWL-RA Develop idioms for supporting Power Types in an OWL-RA context	Develop and apply suite of test cases and test "data" for semantic mappings.	Manage OWL-RA artifacts in RDBMS by integrating and extending vendor's existing XML, RDF and object relational capabilities. Implement Chu Space algorithms in SQL
Results	Foundation for heterogeneous interoperability.	Interoperability across diverse schemas and models using distinct domain semantics expressed in disparate languages.	Scalable interoperability solution easily deployable far and wide.



Note to the reader: A fully worked example of the Stage One approach and its concepts and technology may be found in the Worked Example of First Stage section of this paper.

Destinations of the First Stage

- Support for an open-ended collection of languages used to represent domains and their semantics, e.g., OWL, UML and EDOC employed to model systems and artifacts within the financial management domain
- Select an ontology language to serve as the initial meta language for the framework
    - The chosen ontology language becomes the common meta language for the community of languages, i.e., a common means to:
        - Represent each language's definition
        - Represent language artifacts
        - Manage language artifacts
    - Employ OWL-DL as the "base" ontology language, based on its:
        - Expressivity
        - Decidability
        - Serializations, particularly RDF/XML
        - Web support, e.g., ontology elements as publishable web resources
        - Standardization
        - Community
        - Growing tool support: visual editors, repositories, reasoners
        - Mindshare vis-à-vis the Semantic Web
    - Define OWL-RA, as defined by Von Schweber, as an extension of the base, exhibiting a Relative metamodeling Architecture.
        - OWL-RA is based on the ideas of OWL-FA (Fixed layer metamodeling Architecture) [Horrocks01], [Horrocks03], [Motik05], [Pan2003], [Horrocks05]
            - OWL-FA is decidable and supports metamodeling
            - OWL-FA extends OWL-DL based on the extensions of RDF-FA to correct RDF's non-standard semantics while making it suitable for metamodeling
        - OWL-RA will define compartments for instances, classes, power types [Cardelli87], [Brodnik91], etc. and the interrelationships between compartments, all in a fashion compatible with DL reasoners (based on but relativizing OWL-FA's fixed layered architecture)
        - Benefits include:
            - Decidability (compared with an OWL-Full approach to support meta modeling)
            - Standard semantics (compared with the non-standard semantics of OWL-Full and RDF)
            - Support for metamodeling without the straightjacket of a linear metalevel *stack*; an arbitrary modeling

element may be related to multiple meta constructs without these comprising a linear stack.

- Define an OWL-RA idiom to accommodate many-to-many mapping of terms to classes/properties, thereby achieving a separation of concerns between terminology and semantics, cleanly allowing for synonyms and homonyms.
  - Idioms are language patterns that create a dialect of use
    - For example, Sergey Melnik and Stephan Decker present six independent RDF graphs that represent the commonly held (but non factual) belief that Mozart composed the Requiem with the assistance of Salieri [Melnik2004]. We see each of their representations employing a distinct language idiom that may be objectified and made explicit.
    - Each language idiom may be represented as an ontological template that does not require quantification over classes and therefore avoids the need for higher order logic [Goguen 2005, private communication]

- Define each community language using the framework's meta language
  - Define Semantic Core using the framework's meta language; this produces an ontology of Semantic Core, or more generally, a "theory" of Semantic Core.
  - Define each community language using the framework's meta language. This produces an ontology of each language; more generally we refer to each product as a "theory" of its respective language.
  - Obtain ontology class and property instances for each language theory:
    - For a language theory (a theory that represents a language) an instance (also called a token) of the theory is a sign (in the sense of Pierce's triadic semiotics) for a domain concept, e.g., Unit Price; it is not a language's representation of the domain concept, e.g., a UML attribute or OWL-DL class representing Unit Price.
    - For a domain theory (a theory of a specific domain) an instance is a sign for a domain entity or relationship; it is not an OWL instance or a UML data element, etc.
  - Derive an IF-classification from each instantiated ontology (where an instantiated ontology may represent a domain theory or a language theory). This idea first appeared in IF-MAP (see below).
- Derive mappings between community languages and also between domain ontologies expressed in community languages

- Represent each ontology mapping (between pairs of ontologies) as an Information Flow (IF) [Barwise97] infomorphism between their respective derived IF classifications. (See Appendix I - Information Flow (IF) proto-primer.)
    - The idea of merging ontologies in a bottom-up fashion based on ontology instances appeared in FCA-Merge [Stumme01].
    - FCA-Merge employs FCA'S (Formal Concept Analysis) formal context, a construct essentially equivalent to a classification of Information Flow or a Chu Space. FCA–Merge then forms a Galois lattice over a merged formal context. In comparison, the method we employ in Stage 1 of the roadmap innovates and extends upon FCA-Merge in two ways. First, we represent the merged Chu Spaces as relations in a relational database management system, enabling the mappings to be derived by SQL stored procedures directly within the database, thus enabling wide-spread application of our approach. Second, we apply the method recursively: (i) first to derive a language map, and then within the context of the "merged" language (as defined by the language map) we (ii) derive a domain map. Thus the method we employ may be used to align, merge and map distinct ontologies expressed in distinct ontology languages. Further, our recursive application methodology may be continued, e.g., to align distinct schemas/models grounded within distinct ontologies expressed with distinct language idioms of distinct languages. This makes our approach applicable to the general class of interoperability scenarios encountered in the real world.
    - The idea of representing ontologies as IF Classifications and ontology alignment/merge mappings as IF infomorphisms first appeared in IF-MAP [Kalfoglou05].
    - The IF-MAP approach extends the applicability and correctness of the FCA-Merge method to mapping situations where common instances may be unavailable, such as is often encountered when aligning a local or "spoke" ontology with a reference or "hub" ontology as hub ontologies frequently lack instances or instance data. IF-MAP, as it is an application of Information Flow to the problem of mapping ontologies, treats each ontology as a local logic. The local theory of a local logic may be used to derive "formal instances" that may then be utilized in the alignment process.
    - The methodology of IF-MAP employs a generate-and-test approach and recognizes that ontologies have two overall groupings of elements: classes and relations. From a very

high level the IF-MAP algorithm first generates a candidate infomorphism between local and reference logics, applies the candidate infomorphism to the relations of the local and reference logics, and if this produces valid results then applies the same infomorphism to the classes of the local and reference ontologies and evaluates the mapping. Eventually the generate-and-test loop will identify one or more candidate infomorphisms that both respect the local and reference classifications and respect each one's constraints (the respective local theory). A slightly more detailed description of the IF-MAP algorithm follows.

IF-MAP Algorithm

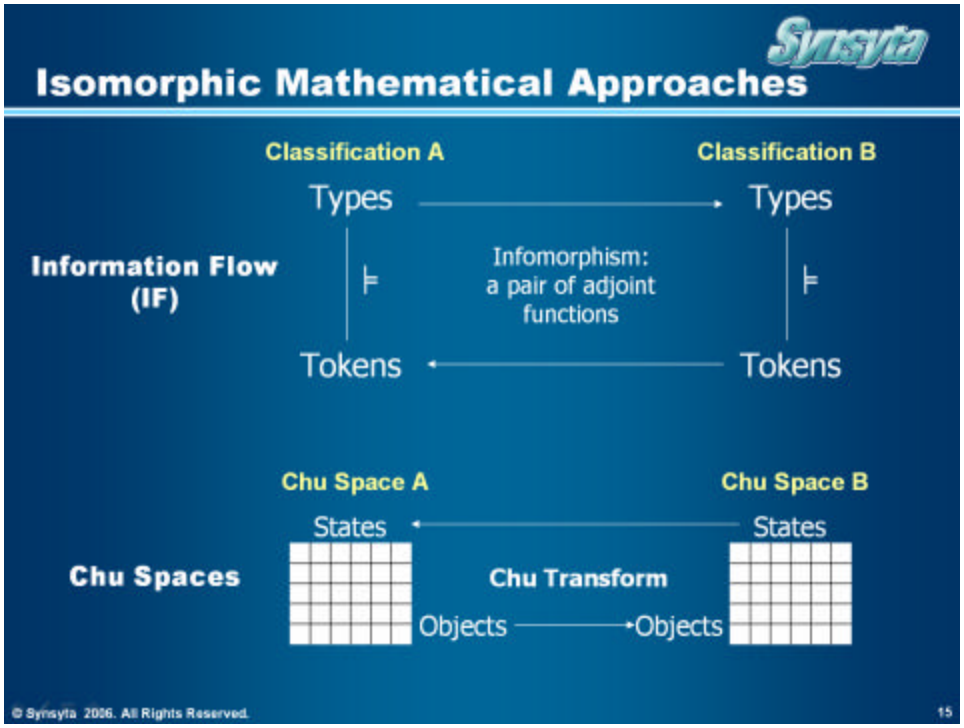
1. Begin a "generate and test" loop.
    a. First attend to mapping relations.
        i. Generate a candidate infomorphism and apply it between a reference relation and a local relation (this generate step is a random selection from the classification of local instances according to a local relation).
        ii. Identify the consequence of the candidate infomorphism on arities (i.e., on the arguments of the relations, implied by the candidate infomorphism applied to relations).
        iii. Repeat this for the remaining relations until all relations of the reference and local ontologies are mapped.
        iv. Consider the infomorphism arrived at by this process as a provisional infomorphism (there may be other possible infomorphisms and we do not yet know this is the best one to use)
    b. Now apply the provisional infomorphism to ontology classes and instances.
        i. Apply the provisional infomorphism in order to classify instances of the local ontology according to classes of the reference ontology.
        ii. Identify formal instances of the reference ontology whose classification (according to classes of the reference ontology) is identical to the classification of local instances (according to the classes of the reference ontology). The "connections" so identified determine the token map of local instances to reference (formal) instances. Furthermore, these connections respect the logic infomorphism between local theories because the formal reference instances "carry" the regular theory of the reference ontology.
        iii. If this identification fails or cannot be achieved then discard the provisional infomorphism and return to step 1.a.i, randomly select another infomorphism, and continue the process from there.
        iv. With success promote the provisional infomorphism to working "logic infomorphism" and exit the generate and test loop.

- The IF-MAP algorithm, while not computationally efficient (it has an order of complexity exceeding o(nlogn)), is applicable to smaller ontologies, hence its utility in a progressive alignment process (see Stage 2 of the roadmap for a discussion of progressive mapping).

o Represent each IF-classification as a Chu Space [Pratt05a], [Pratt99].

- A Chu space may be represented as a simple table of rows and columns
- Represented as a tabular data structure, each Chu Space may thus be managed and processed as an array in a programming language; a matrix, a relational database table, a spreadsheet, etc.
- Employ a workaround if the resulting Chu Space is not biextensional, e.g., add a column to distinguish otherwise duplicate rows; add a row to distinguish otherwise duplicate columns

o Represent each infomorphism (between pairs of classifications) as a Chu Transform.



o Represent each Chu Transform as a Chu Space

- Following all of these prescriptions means that a language, a domain ontology, a language mapping and a domain mapping may each be represented and processed as a Chu Space using a tabular data structure.
- In this way languages, ontologies and mappings may all be represented and processed using the same infrastructure, i.e., tooling and methodology.
- To represent a Chu Transform as a Chu Space requires that *the source Chu Space be extensional, i.e., no duplicate rows, and the target Chu Space be separable, i.e., no duplicate columns*. See workaround (above) when needed.

        - (See Appendix II – Representing Chu Transforms as Chu Spaces.)
- Execute language and ontology definitions and mappings
    - Manage all Chu Spaces as relations in an RDBMS
        - An abstract syntax, including a relational schema, is defined by which Chu Spaces may be embodied and processed as relations.
        - A relational schema is developed or otherwise obtained to manage artifacts expressed in the framework's meta language (i.e., OWL-RA)
        - A set of "model to text" mappings between the abstract syntax of the framework and the world of surface syntaxes is defined and supported by the facilities of the DBMS [Wigetman06]
            - Standardize on XML representations for surface syntaxes supported by an open-ended collection of XSDs (XML Schema Definitions)
            - Employ DBMS load operations to transform each XML surface syntax into the abstract syntax of DBMS-resident Chu Spaces and ontologies in order to load artifacts into the DBMS
            - Employ DBMS SQL functions and/or stored procedure packages to serialize abstract syntax as XML surface syntaxes in order to export artifact from the DBMS
        - Process relationally-embodied Chu Spaces using SQL, e.g., to navigate Chu Spaces, derive transformations/mappings, apply transformations/mappings, derive the Galois lattice (described in the Worked Example of First Stage section below) over each Chu Space that represents a classification, etc.
    - Obtain/provide common instances for ontologies to be mapped, e.g., aligned and/or merged.
    - Employ a Chu Space embodiment of an "FCA-Merge"-like technique [Wille96], [Stumme01], [Priss05] to map two ontologies having common instances.
        - Employ techniques, e.g., as supported by NeoLogical SURVEYOR [Von Schweber 2004] for knowledge surveying, to implement and extend FCA-Merge techniques to provide a scalable infrastructure.
        - The infomorphism between two ontologies, say A and B, is represented as a classification of one ontology's tokens, say A's, according to the types of the other ontology, say B's (where this infomorphism is a Chu Transform represented as a Chu Space itself). The dual classification that classifies B's

token according to A's types may be recovered from the Chu Transform as Chu Space. [Vaughan Pratt 2005, private communication]

Second Stage – Collaborative Ontologies and Composable Language

	Second Stage		
	Definition	**Derivation**	**Execution**
What	Develop a meta ontology of language concern dimensions to augment Semantic Core. This becomes a starting point for defining *living* languages, e.g., ontology languages that may be functionally extended by composing them with additional language components.	Extend Information Flow, Chu Transforms and lattice methods to support progressive, incremental and collaborative mapping and interoperation. Provide the means for users to find common community "examples", i.e., IF tokens, that they can point to.	Adapt and apply Web 2.0 technologies (AJAX, RSS, P2P tagging, etc.) to implement collaborative semantics and collaborative interoperability, thus enabling semantics to be authored and revised in a web page within the user's work context.
Why	Define, revise and extend language; fosters modularity with consistency; improves reuse and paves the way for additional automation of interoperation and more effective collaboration in future roadmap stages. Circumvent "lowest common denominator" interoperability, i.e., when the consumer's language is not sufficiently expressive to represent knowledge provided by a producer's richer source language.	Realize "continuous improvement" in interoperability while keeping end points loosely coupled. For example, each user and community becomes free to independently evolve their own semantics, as in a user adding meta data to a web form so that they may provide additional, relevant data while relating this new data to existing community data, thus enabling interoperability.	A move to collaborative methods requires light weight, agile implementations, e.g., to incorporate users into the process of developing, revising and mapping their semantics with the community's we cannot insist that the user install and learn thick client ontology tooling. Rather, a user must be able to simply go to a web site that provides all the capabilities they need without breaking the context of their work and their domain.

Second Stage

	Definition	Derivation	Execution
Key tasks	Extend Semantic Core with a meta ontology of language concern dimensions. Extend framework's meta language with composition operators. Explore and harvest e-Connections research.	Develop process for collaborative mapping by applying and extending progressive mapping techniques. Explore the derivation, simplification and application of "formal instances". Develop knowledge surveying application to survey community types and tokens within the user's context.	Develop a collaborative web site using extended AJAX and related Web 2.0 technologies for community semantics.
Results	Users and their communities freed from the constraints of rigid formalisms and centrally dictated mandates can interoperate without information loss.	Progressive, incremental and collaborative mapping rather than mapping, alignment and merging that is all-or-nothing, all-at-once and individual and isolated.	Users and user communities on the knowledge fabric take control of their semantics and meta data without breaking interoperability, i.e., think locally, interoperate globally.

- Collaborative ontology mapping and collaborate-to-interoperate
  - Progressive mapping of language theories and domain theories
    - In Stage 1 we align, merge and generally map ontologies in their entirety and all-at-a-time. While this is an excellent strategy to preserve integrity and consistency it leaves open the question, for example, of how changes may be applied to large ontologies that have already been mapped without breaking the existing mapping.
    - Here in Stage 2 we aim to enable an incremental mapping of ontologies, say for example, as the ontologies change and evolve.
    - The IF-MAP method pioneered by Kalfoglou and Schorlemmer has been extended by its authors to support what they call "progressive" mapping. [Schorlemmer05a]

        - In Stage 2 we embrace and extend progressive mapping methods but implement and operate such in our Chu Space formulation (rather than their native Information Flow form). Part of our extension is to apply progressive alignment to language theories as well as domain theories operating in a merged language context.
    - Collaborative mapping
        - Once the framework has gained support for progressive ontology alignment and mapping we can build upon it to enable collaborative mapping.
        - Collaborative mapping is based on the idea that each individual and each group of individuals (regardless of hierarchic level) likely possesses their own ontology and potentially their own language (at the very least, their own language idioms). This makes for a vast knowledge fabric of ontologies, idioms and languages, with individuals mapped to each other and to the communities of which they are a part, plus communities mapped to other communities and to the subsuming communities of which they are a part.
        - Collaborative mapping is the application of automated, progressive mapping techniques to maintain this "social" knowledge fabric, such that end-to-end interoperation becomes possible across the fabric but *without the tyranny of mandatory global ontologies, idioms and languages* (on any scale).
        - Power is pushed out to the network's edge rather than concentrated (and mandated) at the center. This inextricably brings the user into control of not only their data but also their metadata and potentially the language they use to express knowledge. Upper ontologies become emergent artifacts from the view of collaborative mapping.
        - To so enable users and user communities requires that they be empowered with the necessary tools. But thick client applications, designed for knowledge engineers and developers, that require training, skill and significant compute resources, just won't do.
        - To push the power to the edge requires that we at minimum emulate, if not outright adopt, the Web 2.0 technologies that are leading the charge in social computing. [O'Reilly05] Prime among these are AJAX (Asynchronous Javascript and XML) as it brings rich GUI functionality, like drag and drop, to browser-based applications without the need for plug-ins, applets or the like.

- We must be very conscious when selecting, adopting and applying Web 2.0 technologies, as they commonly make assumptions that are not entirely compatible with the objectives of this effort. For example, AJAX assumes network connectivity to the server in order to function; disconnected work using an AJAX approach is not something one gets “out of the box”.
- The approach we advocate for the use of Web 2.0 technologies involves replicating light-weight server components to the local client machine; this may include a web server and a persistent store. This architectural pattern was applied to 3D computing and VR in [Von Schweber 1998a] and [Von Schweber 1998b]. This marries the benefits of Web 2.0 with disconnected operation, and it gains additional concomitant advantages.
- Specific Web 2.0 products that may be useful towards building collaborative mapping:
    - Tibco General Interface
    - Morfik Javascript Synthesis Technology and WebOS Apps Builder
    - ICESOFT ICEfaces
    - Sun Java Studio Creator
- Using our flexibly-connected variant of Web 2.0 we must enable a user or user community to extend their domain ontology without breaking the context of their work or task.
- For example, a user filling out a web form may be unable to enter relevant data for lack of an appropriate form field.
- Said user should be able to dynamically create a new field and populate it with the relevant data, but in and of itself this is insufficient and hazardous. From the system's point of view the user created field constitutes new meta data of which it knows nothing, potentially a new element of the system's domain ontology.
- There are two overall ways that suggest themselves for handling this situation; each makes a different set of assumptions.
    - If we assume the user has no personal domain ontology then we want to induce the user into educating the system about the newly created field, specifically the domain ontological concept that defines the field. In the spirit of the instance-based mapping approach we advocate in this roadmap we must obtain from the user instances of this concept, ideally "common" instances already known to the

system's domain ontology. This initiates progressive mapping between the newly suggested user concept and the system's existing domain concepts.
        - If we assume the user possesses their own personal domain ontology (with instances) then we initiate progressive mapping between the user's domain ontology and the system's.
    - In both of these cases we employ knowledge surveying, e.g., NeoLogical SURVEYOR, to adaptively identify the most suitable contexts for the progressive mapping. Knowledge surveying may reveal common instances (between user and system) and one or more existing concepts in the system's domain ontology that may be used to define the new field. In any event the user is enabled to provide additional data that becomes understandable to the system and its other users.
    - We recommend exploring the use of personal ontologies (the second scenario above) for each user and user community.
- Construct formal instances for each ontology. This is critically important when an ontology possesses no instances in common with another that it is to be aligned/merged with, as for example with a hub ontology, e.g., Semantic Core.
    - IF-MAP refers to a fundamental theorem of representation of Information Flow that provides the basis for constructing formal instances. This is the Representation Theorem 9.33 of Barwise and Seligman.
    - A set of derived formal instances reflect the theory of the local logic, i.e., they manifest the constraints obeyed by the types of the classification, but they are typically awkward (to look at one it is not at all obvious what they mean out of context, compared to say, a domain token).
    - Explore the use of the structure of Semantic Core to simplify the structure of the formal instances derived from it so that such formal instances are more user friendly. Note: Semantic Core has already simplified the mapping by identifying the core constructs of each supported language (simplified as compared with a simple amalgamation of all constructs from the languages supported)
    - Even when common instances are available, formal instances reify the local, potentially regular theory of the ontology and the infomorphism then respects the local theories, hence it is a logic infomorphism. Thus formal

instances are valuable even for scenarios that posses common instances.

- Composability
  - e-Connections (note e = Epsilon) [Grau04], [Grau05] to tease apart modules and represent the "system" of ontology modules (noting that these ontology modules may represent language semantics or domain semantics).
  - Language concern dimensions populate a meta ontology as language-specific Powertypes, (see Appendix III – Power Types) e.g., for decomposing Semantic Core on a finer scale and supporting Multi Dimensional Separation of Concerns (MDSOC). Key dimensions include:
    - Abstract syntax constructors – the elements of the syntax of a language, including both logical and non-logical symbols.
    - Idioms – the "design patterns" of a language, designating and itemizing the many ways a language may be used (e.g., Melnik and Decker showed six ways that RDF may be used to express the commonly held (but false) belief that Mozart composed the Requiem with the assistance of Salieri). [Melnik2000]
    - Axiomatic Semantics – the axioms (i.e., sentences) that collectively define a language's semantics
    - Proof Theories – the inference patterns supported by a language that produce valid conclusions, e.g., modus ponens.
  - Galois lattice over the classification of language tokens by language types
    - In Stage 1 we form the Galois lattice over a merged Chu Space (we do this for both language and domain mappings).
    - Here in Stage 2 we (additionally) form a Galois lattice over a merged Chu Space where language concern dimensions are types and concern dimension values are tokens.
    - This parallels the Information Flow Framework's (IFF) Lattice-of-Theories (LoT) but at the language level (the IFF LoT operates at what we've been calling the domain level, albeit the "upper" part.
    - This construction sets the stage for the composition of language definition (molecules) from language elements in the spirit of Managed Logic (see Appendix IV – Managed Logic).
  - Moving in the direction of a composable axiomatic semantics
    - Extend each IF classification of an ontology (domain or language) to a full local logic
      - The local logic adds to the classification all of the constraints met by types of the classification.

        - Constraints are represented as sequents.
    - Extend each infomorphism between ontologies (domain or language) to a full logic infomorphism between local logics
        - A logic infomorphism is an infomorphism whose type mapping respects the constraints of the local logics.
- Knowledge surveying employed to survey language features (represented as language concerns) in order to select an existing language, choose components to extend an existing language or compose a new one.

Third Stage – Living, Liaising Hubs

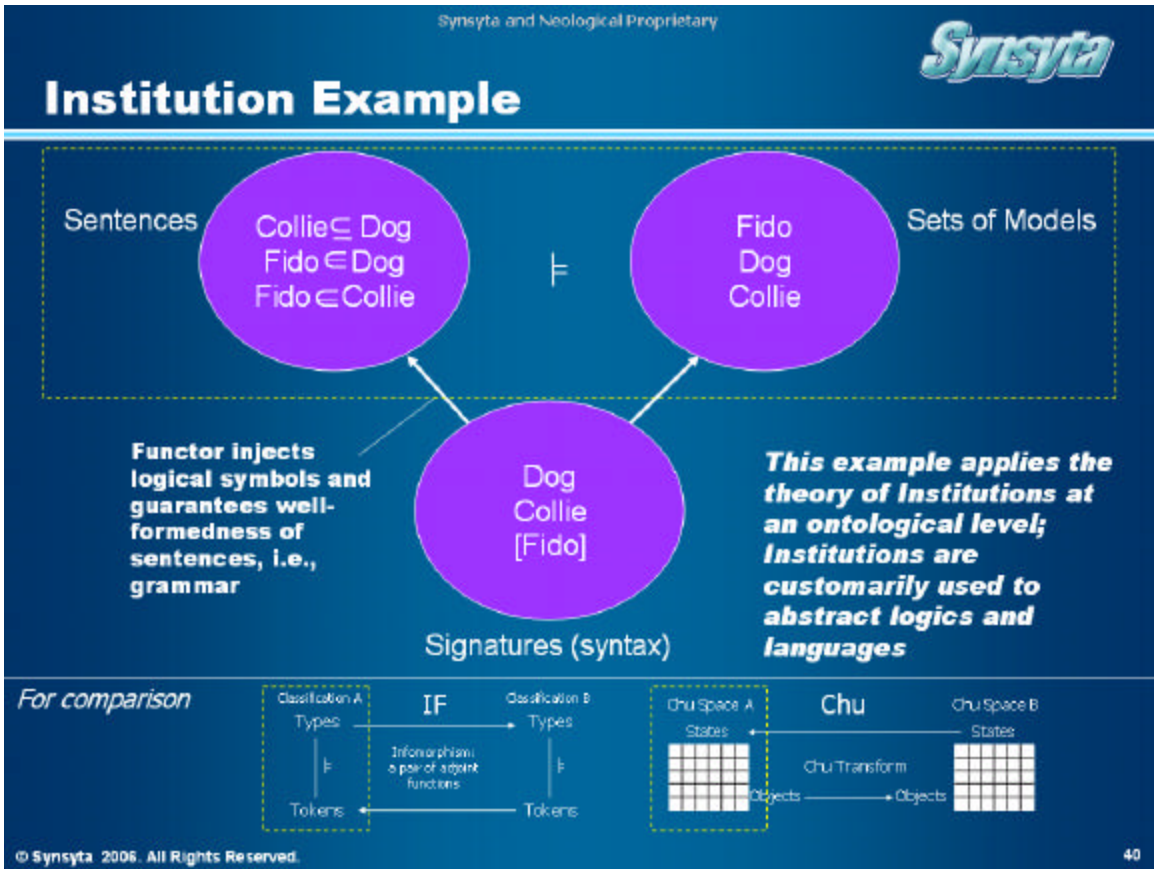
Third Stage			
	Definition	**Derivation**	**Execution**
What	Lift IF-Classifications/Chu Spaces to full Institutions over two phases.	Chu Transforms lifted to Institution Morphisms.	Phase I: Manage each Institution natively but "flatten" to a Chu Space for processing using infrastructure of Stages 1 & 2 Phase II: Extend infrastructure to process Institutions natively.
Why	Explicitly incorporate syntax into the mathematical representation and mapping formalism.	Derive mappings between arbitrarily complex syntaxes.	Exploit infrastructure of Stages 1 & 2 then extend it.
Results	Modularization of syntax and syntax mapping. Composability of language definition, including syntax and semantics.	Greater generality of mapping and semantic interoperability.	Earlier support for complex syntaxes (far beyond XML). Enablement of self-adaptive "living" hubs.

- Lifting of classifications to Institutions
    - Chu Spaces and Chu Transforms are extended to become full-fledged Institutions [Goguen06] and Institution Morphisms (respectively), thus providing a framework for defining languages, logics and ontologies via formal composition. [Schorlemmer05b], [Kent2004], [Voutsadakis05], [Sernadas2005]
        - Local logics extended to become Institutions
            - IF Classifications, local logics and Chu Spaces do not formalize the relationship of their types and tokens to the syntactic context with which they are articulated.
            - An Institution (with regard to the Theory of Institutions) extends the type/token classification structure of Information Flow and Chu Spaces:
                - An explicit set (alternatively a category), called Sign (for signature), explicitly captures the

syntactic context of a classification. Note that Sen can express syntactic elements of arbitrarily complex structure.
        - The set of types of a classification are lifted to a set (or category) of sentences, called Sen, expressed using the syntactic elements of Sign.
        - The set of tokens of the classification are lifted to a set (or category) of models, called Mod.
        - The classification of tokens by types becomes a satisfaction relation between models and sentences as expressed in the syntactic elements of Sign.
        - Category theoretic functors map Sign to Sen and Sign to Mod.
        - An Institution therefore represents the truth semantics of a logical system but in a fashion where the syntax of the logical system is made explicit.
        - Acting between Institutions (as just described above) are Institution Morphisms, the Institutional analog of IF logic infomorphisms and Chu Transforms.
        - A key distinction between an Institution Morphism and its less expressive analogs is that an Institution Morphism can express the mapping of truth semantics under change of notation (syntax), as when moving (mapping) from FOL to HOL or from OWL-DL to FOL.
      - In Stage 3 we introduce a set of richly structured signatures to support arbitrarily complex and relative meta-relations and compartments for Powertypes and other, as yet unanticipated domain semantics.
    - Logic Infomorphisms are extended to become Institution Morphisms
  - Galois-ified Institutions (parallels Institutional IFF LoT)
    - Just as in Stage 1 we form the Galois lattice over a merged Chu Space (for language and domain theories) we may also form the Galois lattice over an Institution or a merged Institution (a merged Institution is a construct similar to a merged Chu Space, a step along the path to obtaining an Institution Morphism). We call this a Galois-ified Institution and its vertices we call G-frames.
    - A G-frame is analogous to a formal concept in FCA (Formal Concept Analysis) but may be obtained over arbitrary logics

(whereas FCA only works over FOL). Each G-frame represents a Galois connection between a set of sentences (from Sen) and a set of models (from Mod). Note: a Galois connection is established when certain closure relations hold.

- Lifting may be accomplished in two phases
  - Phase I – Define and manage an Institution using Institution-specific data structures then pre-process each Institution to flatten to a classification/Chu Space for reasoning using the infrastructure developed during stages one and two of the roadmap.
  - Phase II – Process and reason against native Institution(s), i.e., extend the infrastructure developed during stages one and two to natively manage and process Institutions.
- Institutionalize the meta ontology of language concern dimensions (using a single mathematical system to formalize it, i.e., a single meta language) to support the definition and composition of languages/logics and ontologies/models across a concern space of four dimensions.
- Define a category of theories as an enrichment of the LoT whereby the LoT is a *broad category* (i.e., same set of objects) embedded in the category of theories and an edge of the lattice may represent not only inclusion but a more general morphism. Plus there may be additional morphisms added between the vertices of the lattice.
  - Define Institutional signature to represent linear stack, or spiral, recursive construction to support an arbitrary number of roles, etc.; even to support compartments for Powertypes
  - For this relative meta-structure (with compartmented Powertypes) as defined within the Institution's signature: preprocessing "flattens" or "selects" only two adjacent meta regions (single compartment) before submission to a DL reasoner.
- Upgrade the Institution's set of Sentences to a category of sentences where morphisms between sentences constitute proofs, e.g., from these sentences the morphisms, respecting associativity, derive a new sentence (the conclusion) from the previous sentences (the premises). [Mossakowski2005], [Pfenning91]
  - Use the inference patterns to generate the free category which includes the morphisms as proofs
  - See Categorical Logic in [Mossakowski2005].
  - The more general case is the Category of sets of sentences, e.g., to support Infinitary logic with an infinite number of sentences.
- Explore, compare and contrast Galois-ified Institutions coupled with Grothendieck flattening against fibered, Institutional IFF


Synsyta and Neological Proprietary
Synsyta
Institution Example
Sentences
Collie⊆ Dog
Fido ∈Dog
Fido ∈Collie
⊨
Fido
Dog
Collie
Sets of Models
Functor injects logical symbols and guarantees well-formedness of sentences, i.e., grammar
Dog
Collie
[Fido]
Signatures (syntax)
This example applies the theory of Institutions at an ontological level; Institutions are customarily used to abstract logics and languages
For comparison
Classification A
Types
Tokens
IF
Classification B
Types
Tokens
Chu Space A
States
Objects
Chu
Chu Transform
Chu Space B
States
Objects
© Synsyta 2006. All Rights Reserved.
40

Fourth Stage – Living, Liaising Languages

Destination: Knowledge culture supporting co-evolution and symbiosis

Fourth Stage

	Definition	Derivation
What	Lifting of Institution to Charters and Parchments Model Theoretic Semantics is added as a dimension of concern to the MDSOC Institution supporting decomposing and recomposing Model Theoretic Semantics (brings total number of dimensions to 5).	Institutions are derived from Charters and Parchments, whereby a Parchment is used to generate a Charter and in turn an Institution.
Why	Support for multiple meta-mathematics - opening up single meta mathematics (mathematical system of the framework) to community of meta mathematics that are interoperable with translations (and senses of equivalence) defined between them.	Greater generality and flexibility for framework implementers.
Results	Greater generality. No reliance on a single, common meta mathematics.	Community of interoperable interoperation frameworks, each of which has nearly unlimited freedom of implementation choices.

Potential test cases to use during development of Stage 4

- Add modal operators [(necessary, possible) | (always, sometimes)] with propositional logic as base
    - Need to define how modal operators combine with propositional symbols
    - E.g., Axiom says *always is not sometimes not*
    - This is motivation for explicit operators to add, delete, rename, etc. within the Institution.
- Enriching a certain negation construct in a logic requires a change in the symbol used for negation.
    - Use tilda (~) if it's not an involution and the standard square negation bracket (¬) if it is involution.
    - This can be enforced by a theory morphism, but can it be expressed with only lattice inclusion for a sufficiently rich lattice structure?

Worked Example of First Stage Approach

The worked example is intended to serve several purposes:

- Demonstrate the applicability of several mathematical tools to the problem of interoperability across distinct ontologies and languages within a domain of interest to the readers of this paper.
- Provide accessibility to and understanding of these mathematical tools sufficient to promote appreciation of their relevance, applicability and power.
- Illustrate that such mathematical tools and approaches are not reserved for researchers in ivory towers but rather may be gainfully employed within contemporary software environments, e.g., XML, DBMS and modeling and ontology languages.

It is beyond the scope of this paper to address all the details and complexities of the specific problem we work in the example; indeed, doing so would elide the very principles we wish to illuminate. Following the example we will outline some of the details and complexities to be addressed in a proof of concept.

A note on terminology. Language developers use a variety of terms to designate the types and tokens of their languages. The semantic web community uses the term *ontology* to designate the classes and relations of their languages and the term individuals to designate the tokens. The Object Management Group employs the term *model* when referring to the classes and relations of UML and MOF. And *map* refers to these same kinds of entities for the topic maps community (and there are many more of such communities, each with their own terminology). Hereinafter we will use the term *theory* to refer to ontologies, models, maps and the like. Our justification for use of the term theory is that ontologies, models, maps and the like are all systems of constraints, equivalencies and implications over a universe of discourse; it has also been shown that each of these may be represented as a theory in a suitable logic. Furthermore, theories may be about anything; to avoid confusion we will distinguish between a theory of a language, say of OWL-DL, versus a theory of a domain, say of receivables accounting.

Worked Example part one – Language Map

1) Select two languages to map, a source and a target. For this example we will use UML Class and OWL-DL.
2) Select a meta language: we use RDF for the example (more generally, OWL-DL or OWL-FA).
3) Use the meta language to define the language constructs of the source language. For this example we define an RDF theory of UML class modeling.

a. The source language theory should include a number of language instances. In this usage a language instance is a sign that designates a concept of a domain entity or relationship, such as a purchase order line item, a line item attribute, the relationships that connect a line item with each of its attributes, etc.

First we assert that that UML class and attribute are "things" and represented as RDFS classes. We also assert that a UML class "is related to" a UML attribute; we represent "is related to" by defining an RDF property. For clarity we omit namespaces in the RDF markup of our example.

```
<rdfs:Class rdf:about="UML_Class">
        <rdfs:subClassOf rdf:resource="thing"/>
</rdfs:Class>

<rdfs:Class rdf:about="UML_Attribute">
        <rdfs:subClassOf rdf:resource="thing"/>
</rdfs:Class>

<rdf:Property rdf:about="is related to">
        <rdfs:domain rdf:resource="UML_Class"/>
        <rdfs:range rdf:resource="UML_Attribute"/>
</rdf:Property>
```

Next we assert that certain domain concepts may be represented by UML class and attribute constructs and assert what owns what.

```
<rdf:Description rdf:about="PO">
         <rdf:type rdf:resource="UML_Class"/>
</rdf:Description>

<rdf:Description rdf:about="POLineItem">
         <rdf:type rdf:resource="UML_Class"/>
</rdf:Description>

<rdf:Description rdf:about="POLineItem_Attribute">
         <rdf:type rdf:resource="UML_Attribute"/>
</rdf:Description>

<rdf:Statement>
        <rdf:subject rdf:resource="POLineItem"/>
        <rdf:predicate rdf:resource="is_related_to"/>
        <rdf:object rdf:resource="POLineItem_Attribute"/>
</rdf:Statement>
```

4) Use the meta language to define the language constructs of the target language. For the example we create an RDF theory of OWL-DL; to control scope we only model the class and property constructs of the target language

a. The target language theory should include the same instances as were used in defining the source language theory.

First we assert that OWL-DL class and property are things and are represented by RDFS classes. We also assert that one OWL-DL class is related to another OWL-DL class by an OWL-DL property; we represent "is related to" by defining an RDF property.

```
<rdfs:Class rdf:about="OWL-DL_Class">
        <rdfs:subClassOf rdf:resource="thing"/>
</rdfs:Class>

<rdfs:Class rdf:about="OWL-DL_Property">
        <rdfs:subClassOf rdf:resource="thing"/>
</rdfs:Class>

<rdf:Property rdf:about="is_related_to">
        <rdfs:domain rdf:resource="OWL-DL_Class"/>
        <rdfs:range rdf:resource="OWL-DL_Class"/>
</rdf:Property>
```

Next we assert that certain domain concepts (the common instances of this part of the example) may be represented by OWL-DL classes and what is related to what.

```
<rdf:Description rdf:about="PO">
         <rdf:type rdf:resource="OWL-DL_Class"/>
</rdf:Description>

<rdf:Description rdf:about="POLineItem">
         <rdf:type rdf:resource="OWL-DL_Class"/>
</rdf:Description>

<rdf:Description rdf:about="POLineItem_Attribute">
         <rdf:type rdf:resource="OWL-DL_Class"/>
</rdf:Description>

<rdf:Statement>
        <rdf:subject rdf:resource="POLineItem"/>
        <rdf:predicate rdf:resource="is_related_to"/>
        <rdf:object rdf:resource="POLineItem_Attribute"/>
</rdf:Statement>
```

5) Define a relational schema for the meta language. The relational schema will be used to manage and manipulate RDF theories, i.e., actual ontology classes, properties and ontology instances. Oracle 10gR2 includes a schema for RDF ontologies within Oracle Spatial; for the mapping process we can use the Oracle schema for RDF.
6) Load the RDF theories of source and target languages into the RDBMS using the relational schema for RDF.

7) Interim summary: At this point in the process we have two languages defined using a common meta language that "classifies" common instances.
8) For the purpose of the language mapping we treat as types the constructs of source and target languages and as tokens the common instances. For this example the class and attribute constructs of UML and the class and property constructs of OWL-DL constitute types. The name of source and target types are "indexed" as needed to guarantee uniqueness and prevent name collisions, e.g., UML_Class for UML class and OWL-DL_Class for OWL-DL class rather than simply "Class" for each (which would indeed cause a collision).
9) Each language's types and tokens, plus its relationship between its types and tokens, constitutes a classification (in the vernacular of Barwise and Seligman's Information Flow). A classification captures the semantics of the language to an extent determined by the set of known tokens; new, previously unseen tokens may change the semantics (the semantics of a classification is not omniscient).
10) We represent each classification (source and target) as a separate Chu Space, a tabular structure of rows and columns. Tokens are represented as rows while types are represented as columns. The intersection of a row and a column is set to equal "1" if that token is of that type and is set to equal "0" otherwise.

Source Chu Space	**UML_Class**	**UML_Attribute**	**U_is_related_to**
PO	1	0	0
POLineItem	1	0	0
POLineItem_Attribute	0	1	0
POLineItem is_related_to POLineItem_Attribute	0	0	1

Target Chu Space	**OWL-DL_Class**	**OWL-DL_Property**	**O_is_related_to**
PO	1	0	0
POLineItem	1	0	0
POLineItem_Attribute	1	0	0
POLineItem is_related_to POLineItem_Attribute	0	1	1

11) We merge the source and target Chu Spaces, thus representing a merged classification. The number of rows of the merged Chu Space is equal to the number of rows of the source Chu Space (and also equal to the number of rows of the target Chu Space) since the tokens are common

between the two Chu Spaces. The number of columns is equal to the sum of the number of columns of source plus target Chu Spaces as the types are, a priori, distinct.

Merged Chu Space	UML_ Class	UML_ Attribute	U_is_ related_to	OWL-DL_ Class	OWL-DL_ Property	O_is_ related_to
PO	1	0	0	1	0	0
POLineItem	1	0	0	1	0	0
POLineItem_Attribute	0	1	0	1	0	0
POLineItem is related to POLineItem_Attribute	0	0	1	0	1	1

12) Form the Galois lattice over the merged Chu Space.
    a. The Galois lattice is a concept lattice; its vertices represent formal concepts over the merged Chu Spaces.
    b. Each formal concept represents a Galois connection between the formal concept's intent (set of types) and the formal concept's extent (the set of tokens).
    c. The Galois lattice represents the "global" classification over the Information Flow channel defined by the infomorphism between the two "local" (source and target) classifications.
    d. The vertices of the Galois lattice are therefore the types of the global classification.
    e. The set of types that constitute each formal concept's intent represent the language constructs that are merged by the language mapping as based on the classification of common tokens according to source and target types.

Note: We carry on the following operations using a relational schema for representing and processing Galois lattices over Chu Spaces. The schema employs three relations: A master relation represents each lattice vertex as a tuple; a detail relation represents each type (Chu column entity) as a tuple; and another detail relation represents each token (Chu row entity) as a tuple. We do not show these embodiment details here but do show the functional effect.

To form the Galois lattice from the merged Chu Space we first merge identical columns, i.e., columns that have the same values in every row.

Source Chu Space	UML_Class	UML_Attribute	U_is_related_to OWL-DL_Property O_is_related_to	OWL-DL_Class
PO	1	0	0	1
POLineItem	1	0	0	1
POLineItem_Attribute	0	1	0	1
POLineItem is related to POLineItem_Attribute	0	0	1	0

Next we merge identical rows, i.e., rows that have the same value in every column.

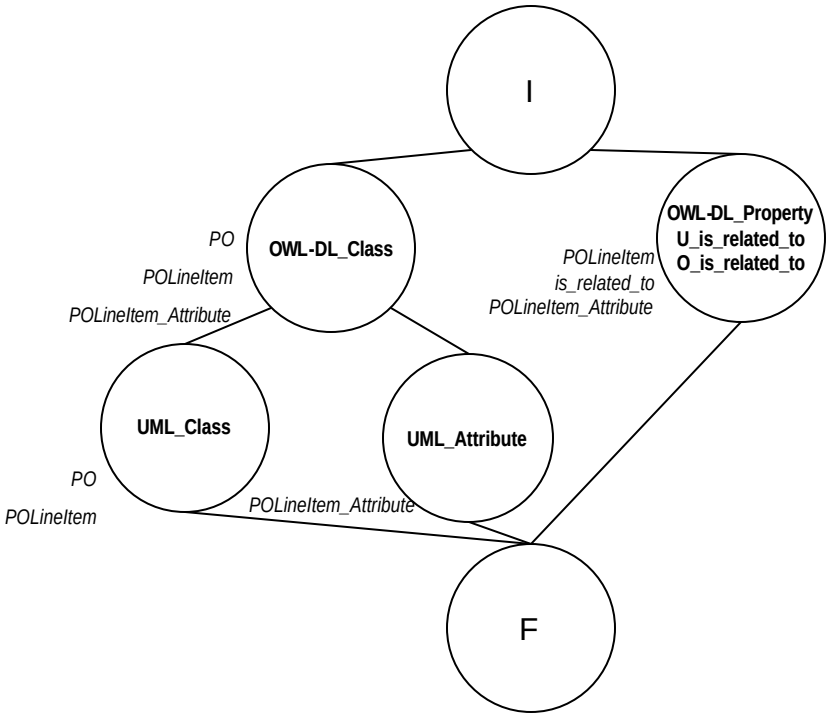
Source Chu Space	UML_Class	UML_Attribute	U_is_related_to OWL-DL_Property O_is_related_to	OWL-DL_Class
PO **POLineItem**	1	0	0	1
POLineItem_Attribute	0	1	0	1
POLineItem is related to **POLineItem_Attribute**	0	0	1	0

Then we order the columns from left to right in descending order by total number of 1's (arithmetic, not logical sum).

Source Chu Space	OWL-DL_Class	UML_Class	UML_Attribute	U_is_related_to OWL-DL_Property O_is_related_to
PO **POLineItem**	1	1	0	0
POLineItem_Attribute	1	0	1	0
POLineItem is related to **POLineItem_Attribute**	0	0	0	1

Not counting the top and bottom vertices we have four vertices in the Galois lattice over the merged Chu Space, one for each column. The Hasse diagram for the lattice is illustrated in the figure below.

Hasse diagram of the Galois lattice over the merged Chu Space



13) Recover the type map (one half of the IF infomorphism) from the intent of the formal concepts of the Galois lattice. This type map ***is*** the language map.

We use the name CLT^{up} to represent the portion of the Chu Language Transform that acts on types, mapping types of the source to types of the target.

Language Map

CLT^{up} : UML_Class → OWL-DL_Class

CLT^{up} : UML_Attribute → OWL-DL_Class

CLT^{up} : U_is_related_to(UML_Class, UML_Attribute) → O_related_to(OWL-DL_Class, OWL-DL_Class) {where type(O_related_to) = OWL-DL_Property

We may now represent this mapping for arbitrary tokens of source and target types (as contrasted with the common instances we employed to derive the

mapping). We first generalize the tokens in each of the source and target Chu Spaces.

Generalized source Chu Space for UML class	UML_Class	UML_Attribute	owns
S1, i.e., a UML class	1	0	0
S2, i.e., a UML attribute	0	1	0
S3, i.e., the owning of a UML attribute by a UML class	0	0	1

Generalized target Chu Space for OWL-DL class & property	OWL-DL_Class	OWL-DL_Property
T1, i.e., an OWL-DL ontology class	1	0
T2, i.e., an OWL-DL property	0	1

Next we represent the Chu Transform (that acts between the source and target spaces) *as a Chu Space*. We do this by classifying the tokens of the source according to the types of the target. However, to accomplish this without ambiguity the source Chu Space must be extensional, i.e., no duplicate rows, and the target must be separable, i.e., no duplicate columns.

Generalized Chu Transform UML to OWL-DL (as Chu Space)	OWL-DL_Class	OWL-DL_Property
S1, i.e., a UML class	1	0
S2, i.e., a UML attribute	1	0
S3, i.e., the owning of a UML attribute by a UML class	0	1

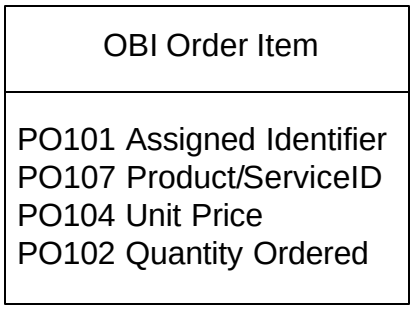
We can recover the language type map, CLT^{up} , from this Chu Transform as Chu Space (TS, or Transform Space, for short) as follows. For each token of the source, say S1, we find it in the TS and identify which columns have equal values. For S1 UML_Class in the source space has equivalent values to OWL-DL_Class in the TS; hence the transform takes UML_Class into OWL-DL_Class. The procedure is repeated for the remaining source tokens, S2 and S3, resulting in transforming UML_Attribute into OWL-DL_Class and the owning of a UML_Attribute by a UML_Class into an OWL-DL_Property.

Note: In general, given an extensional source space, a separable target space and a Chu Transform represented as a Chu Space itself, it is also possible to recover the contravariant map, i.e., how tokens of the target map back to tokens of the source. Were we to do this here, in this particular example, we'd find there are two possible ways in which a specific OWL-DL class can be mapped to a specific UML construct: it can be mapped to a specific UML class or a specific UML attribute, with an OWL-DL property mapping to an owing relationship between a specific UML class and a specific UML attribute. To make this functional, i.e., resolve the choice, we would need to consider a wider array of tokens, consider additional UML language constructs as types (e.g., association) and potentially narrow the target from OWL-DL to OWL-DL using one or more specific language idioms, i.e., design patterns for language usage.

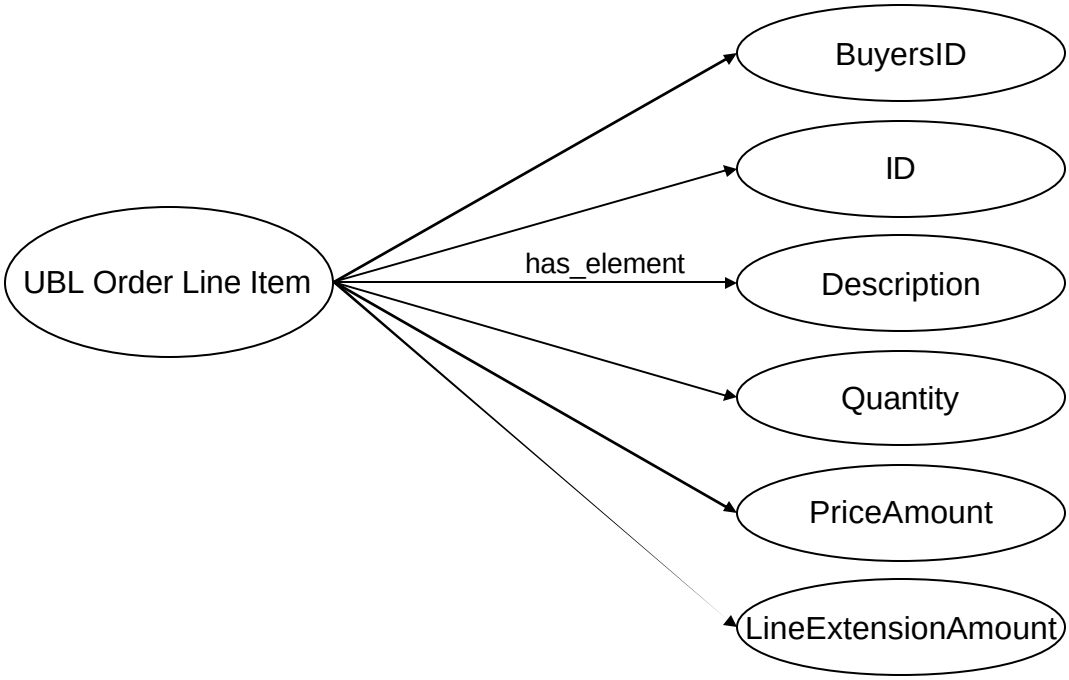
Worked Example part two – Domain Map

1) The domain mapping process recapitulates the very same process used for language mapping but does so applied to domain types and tokens *within the context of the language map*. This last notion is central and critical.
2) Select a source domain theory and a target domain theory; these must be expressed in a language pair for which a language map has already been derived. Of course the source and target domain theories may be expressed in the same language, in which case an identity map exists between the source and target languages. For our example we choose constructs from the domain of accounting, specifically purchase orders line items, expressed in UML Class for the source and a different conception of purchase orders line items for the target, expressed in OWL-DL. For the former we employ RosettaNet's Open Buying on Internet (OBI) community semantics; for the latter we use the Universal Business Language (UBL).

UML class model of RosettaNet OBI (Open Buying on Internet) Order Item



OWL-DL ontology of UBL (Universal Business Language) Order Line Item



3) Source and target domain theories should possess common instances; here "instance" means a sign for a thing or concrete relationship in the domain of discourse. Common instances will comprise the tokens of the source and target domain classifications. For our example here we work with just a single purchase order line item instance; conventionally there would be many common instances.

UML instance of RosettaNet OBI (Open Buying on Internet) Order Item

PO1138_1 : OBI Order Item
PO101 Assigned Identifier = Cust_123 PO107 Product/ServiceID = SKU_abc PO104 Unit Price = $10 PO102 Quantity Ordered = 5 units

Common instances of UBL Order Line Item expressed with OWL-DL

Note: To simplify the markup for this example all properties are represented as object properties; "production" markup would also employ OWL-DL data type properties, e.g., to represent values for quantity and price amount.

```
<rdf:Description rdf:about="PO1138_1">
    <rdf:type rdf:resource="UBL_Order_Line_Item"/>
</rdf:Description>

<rdf:Description rdf:about="Cust_123">
    <rdf:type rdf:resource="BuyersID"/>
</rdf:Description>

<rdf:Description rdf:about="SKU_abc">
    <rdf:type rdf:resource="ID"/>
</rdf:Description>

<rdf:Description rdf:about="$10">
    <rdf:type rdf:resource="Quantity"/>
</rdf:Description>

<rdf:Description rdf:about="5 units">
    <rdf:type rdf:resource="PriceAmount"/>
</rdf:Description>

<UBLOrder_Line_Item rdf:about="PO1138_1>
   <has_element rdf:about="Cust_123"/>
   <has_element rdf:about="SKU_abc"/>
   <has_element rdf:about="$10"/>
   <has_element rdf:about="5 units"/>
</UBLOrder_Line_Item>
```

4) Partition the domain map into component maps according to the types of the global classification (as represented by the vertices of the Galois lattice formed over the merged language Chu Space – see Worked Example part one – Language Map). Each "global" type constitutes a shared language construct across the two languages; it is within each global type that we will map the two domain theories and their instances, global type by global type. The complete domain map is comprised of all the component maps taken together.
    a. The language map takes UML classes and attributes into OWL-DL class. Therefore one of the global types on the language level is class-attribute/class; it is the first component of the domain map.
    b. The language map takes ownership of UML attributes by UML classes into OWL-DL properties. Therefore the other global type on

 erick@synsyta.com

the language level is ownership/property; it is the second component of the domain map.

5) To derive each domain component map we will derive a pair of classifications, one for the source domain and one for the target domain. For our example we will derive two such maps and therefore four classifications:
    a. A pair of domain classifications for the global language type class-attribute/class; we'll call this the class-attr/class component of the domain map.
    b. A pair of domain classifications for the global language type ownership/property; we'll call this the owns/property component of the domain map.
6) Define a relational schema for the source and target languages.
    a. We assume both source and target languages possess an XML serialization supported by an XML Schema Definition (XSD), which in the year 2006 is a reasonable assumption.
    b. Mainstream relational database management systems provide direct support for loading XML data into the RDBMS as well as serializing relational data as XML. Our example is based on the capabilities of Oracle 10gR2 which can register an arbitrary XSD and use it to create an object-relational schema within the Oracle RDBMS to manage and manipulate XML data that conforms to the registered XSD.
        i. We register the XSD for UML/XMI into Oracle, thus generating an object-relational schema to parse and manage UML models within the Oracle RDBMS.
        ii. We register the XSD for OWL-DL into Oracle, thus generating an object-relational schema to parse and manage OWL-DL ontologies within the Oracle RDBMS.
7) Load the source and target domain theories into the Oracle RDBMS.
    a. Slight modifications of headers in the UML XMI and OWL-DL XML may be required, e.g., to point to the schema that was previously registered and thus the object-relational schema that the XML data will be loaded into.
    b. Load the XMI file for the UML model into Oracle.
    c. Load the XML file for the OWL-DL ontology into Oracle.
8) Interim summary and look ahead: We may now use SQL to define relational joins by which each global language type (from the relationally embodied Galois lattice of part one) is employed to join the merged source and target language constructs with their respective domain theory class-attr/class or owns/properties and domain instances. We use these SQL statements in the next step to derive our four domain classifications.

9) Derive the four domain classifications as Chu Spaces from their respective domain theory. The name of source and target types are indexed as needed to prevent name collisions.
    a. For the class-attr/class component of the domain map we derive:
        i. A Chu Space (classification) of common domain instances as tokens according to UML classes and attributes in the source domain model.
        ii. A Chu Space (classification) of common domain instances as tokens according to OWL-DL classes in the target domain ontology.

Chu Space of UML classes & attributes	OBI Order Item	PO101Assigned Identifier	PO107 Product/ ServiceID	PO104 Unit Price	PO102 Quantity Ordered
PO1138_1	1	0	0	0	0
Cust_123	0	1	0	0	0
SKU_abc	0	0	1	0	0
$10	0	0	0	1	0
5 units	0	0	0	0	1

Chu Space of OWL-DL classes	UBL Order Line Item	BuyersID	ID	Description	Price Amount	Quantity	Line Extension Amount
PO1138_1	1	0	0	0	0	0	0
Cust_123	0	1	0	0	0	0	0
SKU_abc	0	0	1	0	0	0	0
$10	0	0	0	0	1	0	0
5 units	0	0	0	0	0	1	0

    b. For the owns/property component of the domain map we derive:
        i. A Chu Space (classification) of common domain relationship instances as tokens according to the UML class ownership of UML attributes of the source domain model.
        ii. A Chu Space (classification) of common domain relationship instances as tokens according to the OWL-DL properties of the target domain ontology.

Chu Space of UML class –attribute relations	owns
(PO1138_1, Cust_123)	1
(PO1138_1, SKU_abc)	1
(PO1138_1, $10)	1
(PO1138_1, 5 units)	1

 erick@synsyta.com

Chu Space of OWL-DL property instances	has_element
(PO1138_1, Cust_123)	1
(PO1138_1, SKU_abc)	1
(PO1138_1, $10)	1
(PO1138_1, 5 units)	1

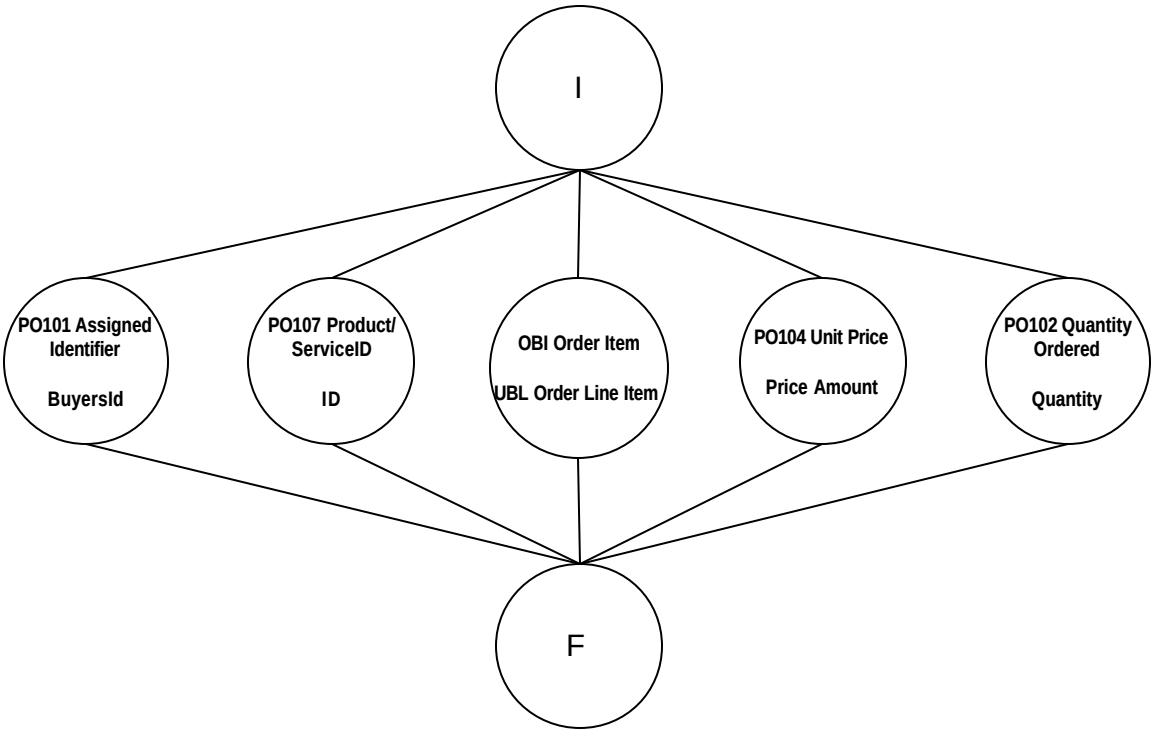
10) Form the merged Chu Space across the domain Chu Space pairs.
    a. Merge the Chu Space pair for the class-attr/class component of the domain map.
    b. Merge the Chu Space pair for the owner/property component of the domain map.
    c. Types for which there are no tokens are dropped from the space.

Merged Class-attr/class Chu Space	OBI Order Item	PO101 Assigned Identifier	PO107 Product/ ServiceID	PO104 Unit Price	PO102 Quantity Ordered	UBL Order Line Item	BuyersID	ID	Price Amount	Quantity
PO1138_1	1	0	0	0	0	1	0	0	0	0
Cust_123	0	1	0	0	0	0	1	0	0	0
SKU_abc	0	0	1	0	0	0	0	1	0	0
$10	0	0	0	1	0	0	0	0	1	0
5 units	0	0	0	0	1	0	0	0	0	1

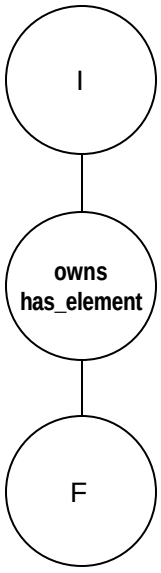
Merged owns/property Chu Space	owns	has_element
(PO1138_1, Cust_123)	1	1
(PO1138_1, SKU_abc)	1	1
(PO1138_1, $10)	1	1
(PO1138_1, 5 units)	1	1

11) Form the Galois lattice over each merged Chu Space.
    a. Each vertex of the Galois lattice over the merged class-attr/class Chu Space is a formal concept whose intent is the set of domain classes to be merged and whose extent is the set of domain instances of the merged class. We will call this the Class-Attribute lattice.
    b. Each vertex of the Galois lattice over the merged owns/property Chu Space is a formal concept whose intent is the set of domain properties or attributes to be merged and whose extent is the set of domain relations of the merged owns/property. We will call this the Relation lattice.

Class-Attribute Lattice



Relation Lattice



12) Recover the type maps (one half of the IF infomorphism) from the intent of the formal concepts of each Galois lattice (class and relation). These type maps together ***are*** the domain map. For the example we have:
    a. A map between specific UML classes and specific OWL-DL classes.
    b. A map between specific UML attributes and specific OWL-DL properties.

We use the name CDT^{up} to represent the portion of the Chu Domain Transform that acts on types, mapping types of the source domain to types of the target domain.

Domain Map

CDT^{up} : OBI Order Item → UBL Order Line Item
CDT^{up} : PO101Assigned Identifier → BuyersID
CDT^{up} : PO107 Product/ServiceID → ID
CDT^{up} : PO102 Quantity Ordered → Quantity
CDT^{up} : PO104 Unit Price → PriceAmount

CDT^{up} : owns → has_element

Note: As we saw in performing the language map (in part one) we may represent each component of the domain map as a Chu Transform as Chu Space. We leave this as a straightforward exercise for the motivated reader.

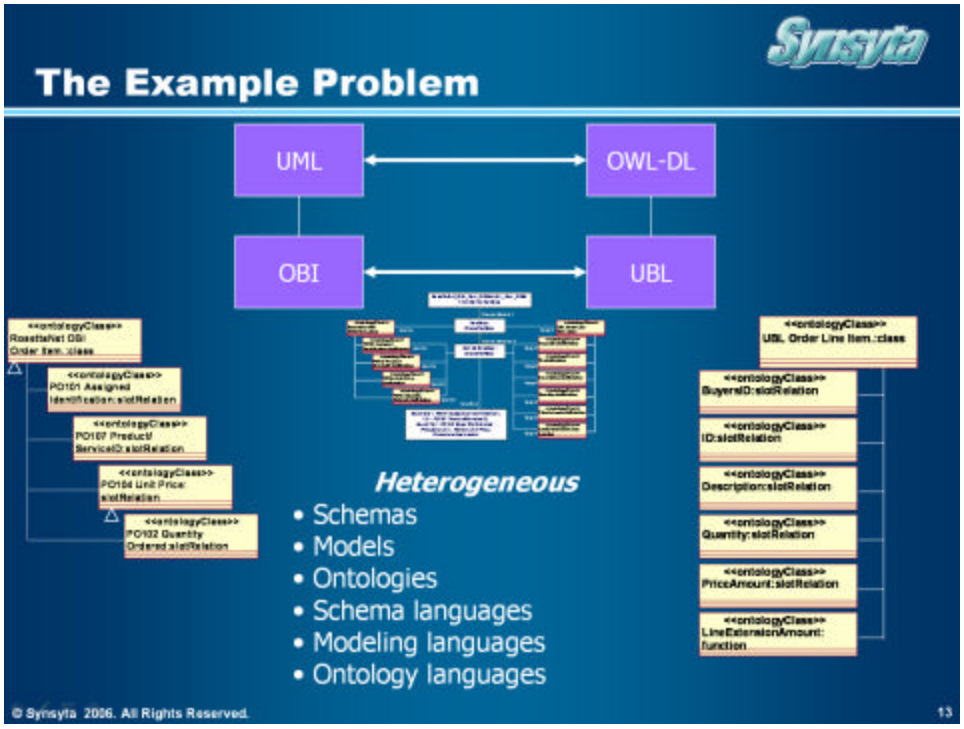
Applied together the language + domain maps may now be used to:

- Transform UML class models into OWL-DL ontologies for management in a semantic web repository, to post on the web, visualize with an ontology editing tool (e.g., Protégé), as content for a web page, semantic markup of a web service, or for input to a reasoning engine.
- Transform OWL-DL ontologies into UML class models for management in a MOF repository, visualization and editing in a UML tool, or to apply within a model driven architecture effort.

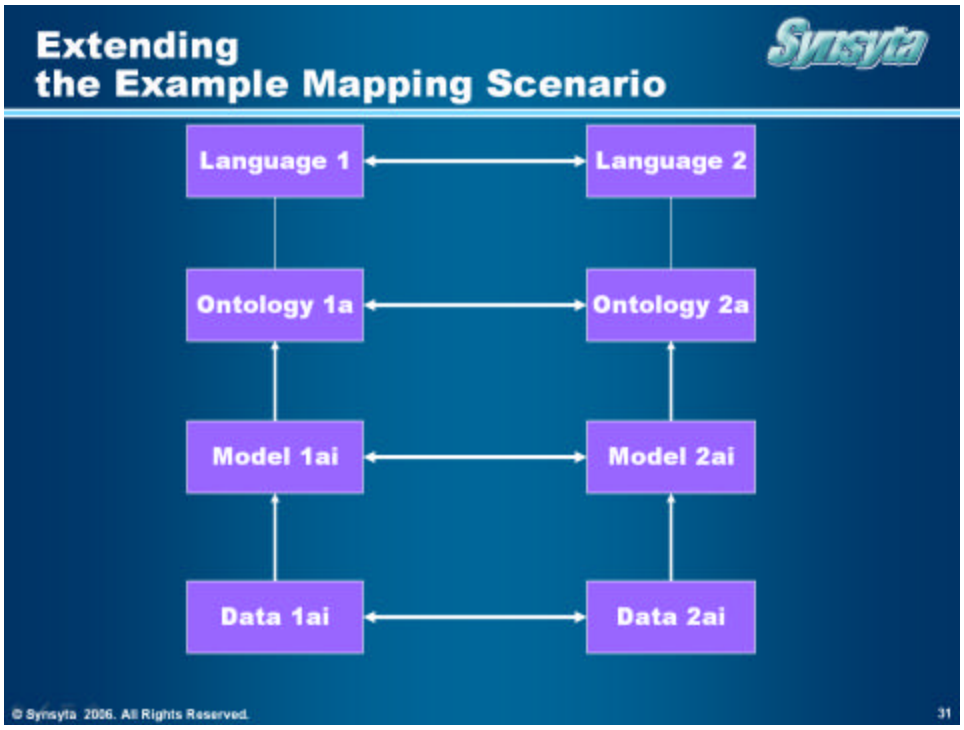
Specific details and complexities not addressed in this example but fully amenable to the approach include:

- UML associations
- UML association cardinality constraits
- UML association navigability
- OWL property > UML association
- OWL property with cardinality constraint < UML association with cardinality = 1
- Pair of OWL properties with inverseof < UML association (undirected)
- OWL property domain/range <> UML association end
- ...

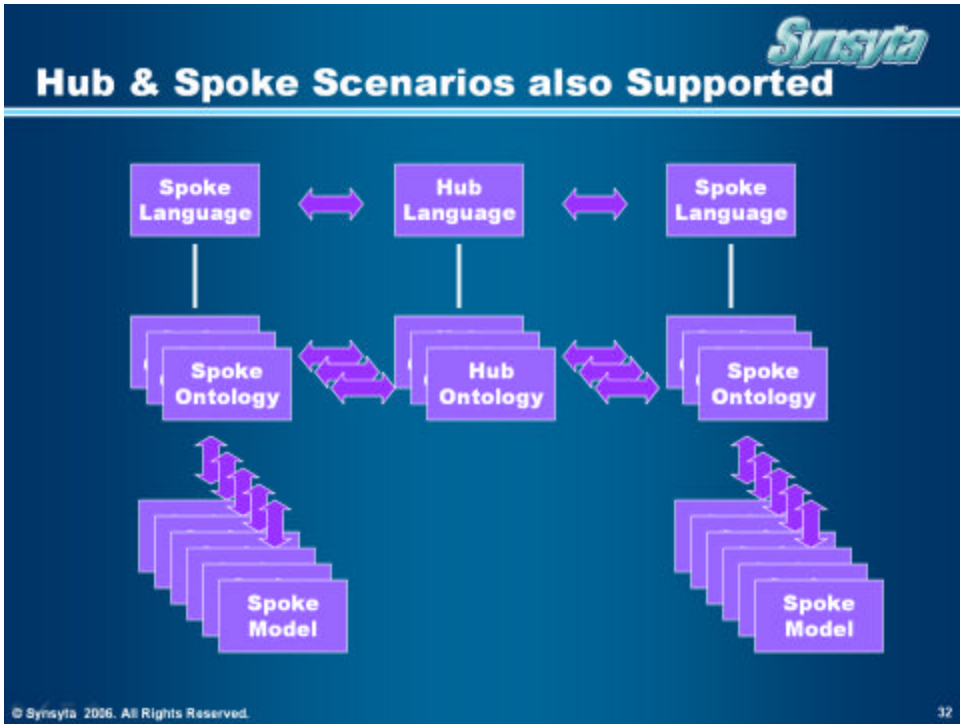
This worked example focused on the mapping of distinct ontologies expressed in distinct languages, as depicted in the figure below.



However the approach has much more general applicability. For example, the same methods may also be used to map schemas/models to the ontologies that semantically ground them, to map such schemas/models to each other across distinct users, systems and communities, and in the final analysis, to transform and migrate data represented by such schemas and models, as depicted in the next figure.



The method may also be applied to hub and spoke architectures as depicted next.



Next Steps: beginning the adventure

With roadmap in hand its time to begin the journey. As it takes time to prepare ground and lay down road surface, we advocate starting with two activities in parallel; one to get moving and produce useful results, the other to prepare for the next stage.

- **First Stage - Proof of Concept**
  - **Run real mapping examples in RDBMS using SURVEYOR and advanced maths**
- **Second Stage - R&D**
  - **Collaborative, progressive ontology mapping using SURVEYOR and Web 2.0 technologies**
    - **Communities of ontologies; emergent "consensus" semantics**
  - **Ontology of language concern dimensions as meta ontology to Semantic Core**
    - **Composable languages**

References

[Barwise97]	John Barwise and Jerry Seligman. **Information Flow**. Cambridge University Press. 1997
[Brodnik91]	Andrej Brodnik and Hemin Xiao. **Typing in Object Oriented Database Systems**. http://www.cs.uwaterloo.ca/research/tr/1991/46/types.pdf
[Cardelli87]	Luca Cardelli. **Structural Subtyping and the Notion of Power Type**. *In Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*. 1987. Available at: http://research.microsoft.com/Users/luca/Papers/StructuralSubtyping.pdf
[Casanave 2006]	Cory Casanave, **Semantic Core in Open Source E-government Reference Architecture (OSERA)** Semantic Core Open Source Enterprise Reference Architecture, http://colab.cim3.net/file/work/Expedition_Workshop/2006_01_24_BootstrappingSOAthroughCOIs/Casanave_OsEra_2006_01_24.ppt Semantic Core http://www.semanticcore.org/
[Goguen06]	Joseph Goguen. **Information Integration in Institutions**. University of California at San Diego. Dept of Computer Science and Engineering. *To appear in **Jon Barwise Memorial Volume** edited by Larry Moss, Indiana University Press*, 2006. http://www.cs.ucsd.edu/~goguen/pps/ifi04.pdf
[Grau04]	Bernardo Cuenca Grau, Bijan Parsia, Evren Sirin, Aditya Kalyanpur. **Modularizing OWL Ontologies**. 2004. http://www.mindswap.org/2004/multipleOnt/papers/modularFinal.pdf
[Grau05]	Bernardo Cuenca Grau. **Combination and Integration of Ontologies on the Semantic Web**. July 6, 2005. http://www.mindswap.org/2004/multipleOnt/papers/Dissertation.pdf
[Horrocks01]	Jeff Z. Pan, and Ian Horrocks. **Metamodeling Architecture of Web Ontology Languages.** *In Proceedings of the First Semantic Web Working Symposium (SWWS'01),* Stanford, July 2001. http://www.cs.man.ac.uk/~horrocks/Publications/download/2001/rdfsfa.pdf
[Horrocks03]	Jeff Z. Pan, and Ian Horrocks. **RDFS(FA) and RDF MT: Two Semantics for RDFS**. Jeff Z. Pan and Ian Horrocks Department of Computer Science, University of Manchester, UK. 2003. http://www.cs.man.ac.uk/~horrocks/Publications/download/2003/PaHo03b.pdf
[Horrocks05]	Jeff Z. Pan, and Ian Horrocks. **OWL FA: A Metamodeling Extension of OWL DL**. *In Proc. of the International workshop on OWL: Experience and Directions (OWL-ED2005).* 2005. http://www.mindswap.org/2005/OWLWorkshop/sub15.pdf
[Kalfoglou05]	Yannis Kalfoglou and Marco Schorlemmer. **IF-Map: An Ontology-Mapping Method based on Information-Flow Theory**. *In Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems. The Netherlands.* 2005 ACM. Also available at: http://www.iiia.csic.es/~marco/ifmap.pdf

[Kent2004]	**Lifting Information Flow to Institutions**. SUO IEE 2004. http://suo.ieee.org/IFF/metalevel/lower/metatheory/synthesis/version20040909.html
[Melnik2000]	A Layered Approach to Information Modeling and Interoperability on the Sergey Melnik and Stefan Decker. **A Layered Approach to Information Modeling and Interoperability on the Web** *Sept. 2004* http://dbpubs.stanford.edu:8090/pub/showDoc.Fulltext?lang=en&doc=2000-30&format=pdf&compression=&name=2000-30.pdf
[Motik05]	Boris Motik. **On the Properties of Metamodeling in OWL**. FZI Research Center for Information Technologies at the University of Karlsruhe. Springer-Verlag 2005. http://dip.semanticweb.org/documents/Boris-Motik-On-the-Properties-of-Metamodeling-in-OWL.pdf
[Mossakowski2005]	Till Mossakowski, Florian Rabe, Valeria De Paiva, Lutz Schroder and Joseph Goguen. **An Institutional View on Categorical Logic and the Curry-Howard-Tait-Isomorphism** *2005* http://www.tzi.de/~till/papers/CurryHoward.pdf
[O'Reilly05]	Tim O'Reilly. **What is Web 2.0. Design Patterns and Business Models for the Next Generation of Software.** Published online by O'Reilley. 09/30/2005 http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html
[Pan2003]	Jeff Z. Pan and Ian Horrocks. **RDFS(FA): A DL-ised Sub-language of RDFS**. *2003* http://www.cs.man.ac.uk/~horrocks/Publications/download/2003/PaHo03c.pdf
[Pfenning91]	Frank Pfenning. **Logic programming in the LF logical framework**. *In Gerard Huet and Gordon Plotkin, editors, Logical Frameworks, pages 149--181. Cambridge University Press, 1991.* http://citeseer.ist.psu.edu/cache/papers/cs/7350/http:zSzzSzwww.cs.cmu.eduzSzafszSzcszSzuserzSzfpzSzpubliczSzelf-paperszSzlfproc91.pdf/pfenning91logic.pdf
[Pratt05a]	**Chu Spaces** published online at http://chu.stanford.edu
[Pratt05b]	Vaughan Pratt. **Attributes as dual types: a unification of presheaves and Chu spaces**. Stanford University. February 25, 2005. http://boole.stanford.edu/pub/aadt.pdf.
[Pratt99]	Vaughan Pratt. **Chu Spaces: Notes for the School on Category Theory and Applications**. Stanford University. July 13-17, 1999. http://chu.stanford.edu/guide.html#coimbra
[Priss05]	Uta Priss. **Establishing Connections between FCA and Relational Databases**. School of Computing Napier Univ. Edinberg, UK. http://www.upriss.org.uk/papers/iccs05.pdf
[Schorlemmer05a]	Marco Schorlemmer and Yannis Kalfoglou. **Progressive Ontology Alignment for Meaning Coordination: An Information Theoretic Foundation.** *Presented at AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.*
[Schorlemmer05b]	Marco Schorlemmer and Yannis Kalfoglou. **A General Theory of Semantic Integration**. *Submitted to Elsevier Science, November 2005.*
[Sernadas2005]	Website of Amilcar Sernadas at Technical University of Lisbon, Portugal. Extensive work on combining logics, including using Theory of Institutions. In particular see FibLog, LogComp and ACL. http://www.cs.math.ist.utl.pt/cs/acs.html

[Stumme01]	Gerd Stumme and Alexander Maedche. **FCA-Merge: Bottom-Up Merging of Ontologies**. In *Proc. 17th Intl. Conf. on Artificial Intelligence (IJCAI '01), Seattle, WA, USA*, pp. 225-230. 2001. http://www.aifb.uni-karlsruhe.de/Publikationen/showPublikation?publ_id=483
[Von Schweber 1998a]	Erick Von Schweber. **SQL3D Café**. *1998* http://www.infomaniacs.com/SQL3D/SQL3D-Cafe.htm
[Von Schweber 1998b]	Erick Von Schweber. **Escape from VRML Island**. 1998 http://www.infomaniacs.com/SQL3D/SQL3D-Escape-From-VRML-Island.htm
[Von Schweber 1998c]	E. Von Schweber and L. Von Schweber. "Computing's Next Wave" Oct 26, 1998. PCWeek (now EWeek) http://www.infomaniacs.com/ComputingFabrics/InfomaniacsComputingFabrics.htm
[Von Schweber 2004]	Erick Von Schweber and Linda Von Schweber. **Neological SURVEYOR 2004**. www.neological.com
[Voutsadakis05]	George Voutsadakis. **Remarks on Classifications and Adjunctions** School of Mathematics and Computer Science. Lake Superior State University. February 2, 2005. http://math.unipa.it/~circmat/vout.pdf
[Wigetman06]	Robert Wigetman and Jurgen Moortgat. **Know Your UML with XML.** Oracle Magazine. January/February 2006 (Pg 67=70) http://www.oracle.com/technology/oramag/oracle/06-jan/o16xml.html
[Wille96]	Bernhard Ganter and Rudolf Wille. **Formal Concept Analysis – Mathematical Foundations**. Springer. 1996

Appendices

Appendix I - Information Flow (IF) proto-primer

The coinage of IF consists of tokens and types, but these constructs are sometimes used in an unconventional manner. Unlike many logical systems, IF applications can (and periodically do) treat a class-like thing as a token and an instance-like thing as a type. That can lead to confusion on initial encounters with IF.

IF emphasizes two key relationships amongst tokens and types. One is *classification* of a token by a type, e.g., the token Lassie is classified as of type Collie. The other relationship is *consequence* between types, e.g., type Dog is a consequence of type Collie in that any token that is classified of type Collie must necessarily be classified of type Dog.

IF packages these two key relationships together into a *local logic*, defined by 1) a set of tokens, 2) a set of types, 3) a classification relation between tokens and types, and 4) the smallest, closed set of constraints on consequence relations (called a local theory, and, with a bit more constraint, a regular theory).

A consequence relation of a local theory in IF is typically represented by a *sequent*. A sequent is written as a comma-delimited string of types that comprise the left hand side of the sequent, a symbol for the consequence relation (+), and another comma-delimited string of types that comprise the right hand side. Here's an example of a sequent: car, convertible + vehicle, muffin. A sequent is "read" in the following way: a token that is of type car AND also of type convertible is by consequence a token of type vehicle OR a token of type muffin. Commas on the left hand side are to be read as AND (conjunctively) while commas on the right hand side are to be read as OR (disjunctively).

The key concept of IF is an *infomorphism*, a pair of maps (functions) that enable *information flow* between two distinct local logics (I'll refer to these local logics as L1 and L2). One of the maps takes types of L1 into types of L2; the other map takes tokens of L2 into tokens of L1 (the two functions are contravariant as they work in opposite directions). Critically and centrally, these two maps, along with each local logic's classification relation, must fulfill a special contract.

> Imagine a rectangle: L1 is on the left side and L2 on the right; types occupy the upper vertices and tokens the lower (so the types of L1 are in the upper left while the tokens of L2 are in the lower right). The left (and respectively right) edges represent the classification relations of L1 (and L2 respectively) while the top edge is the type map from L1 to L2 while the bottom edge is the token map from L2 to L1. The two maps – the type map and the token map - must be

adjoint (in order for the pair of functions to be an infomorphism). One path starts in the lower right, from a token tok2 of L2, the token map is applied to it moving up horizontally to the left, taking it to a token tok1 of L1. L1's classification relation then moves us vertically, telling us that tok1 is classified as one of L1's types, call it typ1. That's one path. The other path says that if you start with typ1 in the upper left hand corner of the rectangular diagram, apply the type map to move horizontally to the right, you end up at one of L2's types, we'll call it typ2. For the pair of functions to be adjoint the token we originally started with, namely tok2, must be classified as of type typ2 (moving vertically up the right hand edge).

When you put it all together the infomorphism simply says that the two maps respect each local logic's classification of tokens by types; that's the central tenet of IF. Because each local logic's classification of tokens by types constitutes that local logic's *semantics*, the infomorphism is, in short, a mapping between local semantics that respects each local semantics.

IF also extends the idea of infomorphism into what it calls a *logic infomorphism*: adding to an infomorphism a constraint that the infomorphism must respect each local logic's local theory, e.g., when you apply the type map to the types involved in a consequence relation that holds in L1 the resulting consequence relation must hold in L2. A simple example of this in the context of natural language is that if in English any token of type Siamese is by consequence a token of type cat then under the type map that takes Siamese to Siamois and cat to chat then any token of type Siamois must by consequence be of type chat.

That a local logic includes both a classification of tokens by types plus a local theory (consequence relations) means that IF ties together the two traditional ways of defining semantics: model theoretic semantics and axiomatic semantics and a logic infomorphism constitutes a semantic mapping that respects both.

See [Barwise97].

Appendix II – Representing Chu Transforms as Chu Spaces

Text by Erick Von Schweber and Vaughan Pratt (from private communication Dec. 2005). Example by Erick Von Schweber.

A Chu transform (f,f'):(A,r,X)->(B,s,Y) is standardly representable as a Chu space if and only if A is extensional and B is separable (T_0).

The Chu space standardly representing (f,f') is (A,t,Y) where t(a,y) = r(a,f'(y)) (= s(f(a),y) by adjointness of (f,f')).

Chu Space

(A,r,X)

	x_1	x_2
a_1	1	0
a_2	0	1

Extensional, i.e., no duplicate rows

Chu Transform as Chu Space

(A,t,Y)

	y_1	y_2
a_1	1	0
a_2	0	1

Chu Space

(B,s,Y)

	y_1	y_2
b_1	0	1
b_2	1	0

Separable, i.e., no duplicate columns

For the Chu Transform defined by the adjoint pair of functions f,f' where f: A->B and f': Y->X as:
f(a1) = b2 f(a2) = b1 f'(y1) = x1 f'(y2) = x2
then s(f(a),y) = r(a,f'(y)) for all a in A and y in Y.

For Y = {y1, y2} and A = {a1, a2} we have:
s(f(a1), y1) = r(a1, f'(y1)) => s(b2,y1) = r(a1,x1) => 1 = 1
s(f(a2), y1) = r(a2, f'(y1)) => s(b1, y1) = r(a2, x1) => 0 = 0
s(f(a1), y2) = r(a1, f'(y2)) => s(b2, y2) = r(a1, x2) => 0 = 0
s(f(a2), y2) = r(a2, f'(y2)) => s(b1, y2) = r(a2, x2) => 1 = 1

Now, t(a,y) = r(a,f'(y)) = s(f(a),y) for all a in A and y in Y, thus yielding:
t(a1, y1) = r(a1,f'(y1)) = s(f(a1),y1) => t(a1,y1) = r(a1,x1) = s(b2,y1) => 1 = 1 = 1
t(a1, y2) = r(a1,f'(y2)) = s(f(a1),y2) => t(a1, y2) = r(a1,x2) = s(b2, y2) => 0 = 0 = 0
t(a2, y1) = r(a2,f'(y1)) = s(f(a2),y1) = t(a2, y1) = r(a2, x1) = s(b1,y1) => 0 = 0 = 0
t(a2, y2) = r(a2,f'(y2)) = s(f(a2), y2) => t(a2, y2) = r(a2, x2) = s(b1, y2) => 1 = 1 = 1

f is recovered from (A,t,Y) as the unique f:A->B such that

for each a in A, f(a) = b_a where b is an element of B satisfying s(b_a,y) = t(a,y) for all y in Y; said b exists because (A,t,Y) was obtained from f that way, and is unique by separability of B. f' is recovered dually.

The "only if" follows because

if A is not extensional there are at least two possible choices for f',
while if B is not separable there are at least two possible choices for f.

One could imagine many other ways of representing a Chu transform as a Chu space; the reason for the "standardly" is to restrict to this particular representation. Without some sort of limitation on the representation the "only if" would not hold.

See [Pratt05a], [Prat05b], [Pratt99].

Appendix III – Power Types

Consider the Dog example from Jim Odell. It goes like this. You define a class for Dog. Then you define subclasses of Dog for Collie, Beagle and Spaniel. Fido is an instance of Beagle. Lassie is an instance of Collie. So far so good, all directly representable in a decidable description logic like OWL.

Now we introduce another class called Breed. The problem is that this forces Collie to serve in two roles, as both class and as instance. As before Collie is a class with Lassie as an instance; but Collie must also serve in the role of instance - Collie is an instance of Breed. Either this concept is the same thing that is simultaneously both class and instance, or there are two things, one a class, one an instance, both map to the term "Collie" and an unknown relationship holds between the two. The former approach is incompatible with a description logic; the latter presents a currently unknown and undetermined relationship.

Ian Horrocks and his student Jeff Pan handled collisions between such twin roles (class and instance) in the context of meta modeling by stratifying these roles over distinct meta-levels, i.e., the Dog class plays the role of instance to the language construct Class (at M2) while it plays the role of class to instances like Fido (at M0).

In this case – with Collie and Breed – we do not have this luxury; Breed is clearly at the level of domain ontology, the same level as Collie; yet Breed acts as something much more like a language Class construct despite its being domain-specific. And it's not alone; just consider Kingdom, Phylum, Class, Order, Family, Genus, Species, Race and Haplotype. There may be many constructs that are domain-specific yet have the force (and taxonomic structuring power) of language constructs that we routinely relegate to the domain-independent language layer but are part of an individual's or community's ontology.

The notion of power type was first introduced by Luca Cardelli in the 1980's [Cardelli87]. Cardelli compares a subtype with subsets and a power types with power set (the set of all subsets of a given set). So in terms of our example, if the given type Dog is designated by the set of all dogs, and its subtypes are given by each subset of the set of dogs (e.g., the set of Beagles, the set of Spaniels, etc.) then the set of all subsets of dogs is the power set of dogs and the power type of dogs is the set whose members are these subsets. Hence Beagle, Collie, Spaniel, etc. are members of the power type of Dog; we call this Breed.

Erick Von Schweber Synsyta LLC erick@synsyta.com

Appendix IV – Managed Logic

Concrete example of Managed Logic (MAGIC)

Erick Von Schweber
12-02-2004

Note to the reader: This Managed Logic appendix is a very slight revision of the 12-02-2004 version; a change was made to the introduction to promote clarity. 01-18-2006

Managed Logic was our first roadmap towards automated semantic interoperability and living languages.

Much of what we described on the road to Managed Logic is achieved in Stage 1 of this current Roadmap for Semantics in Netcentric Enterprise Architecture and demonstrated in its worked example above. We include this paper here as an appendix for the reader's convenience.

Table of Contents

  erick@synsyta.com

Introduction

The objective of Managed Logic, MAGIC for short, is to serve as a facility for the definition and evolution of systems of synthetic languages and their artifacts. Key to the MAGIC facility is the capability to automatically derive transformations between language systems from the definitions of such systems within the facility. MAGIC has potentially very broad applicability, across logics, modeling languages, ontology languages, programming languages, message formats, data models, etc.; such generality and horizontal applicability can make it difficult for the newcomer to understand. To promote understanding of the approach taken by MAGIC it was deemed desirable to work a concrete example in a commonly understood domain, e.g., accounting. Below we introduce the problem and the problem setup, including that which is given and that which we aim to achieve.

The challenge of the example to be worked, generally stated, is to transform a relational representation into a semantically equivalent XML representation and accomplish this with as little human involvement as is practicable. More specifically, we are given a relational tuple of data for a line item of a transaction. Also given is a reference to the relation schema that the tuple conforms to. Each relational attribute of the referenced schema itself refers to a relational domain that is grounded in a formal semantic, e.g., a source ontology. The challenge is to transform the provided relational tuple into XML structured data for a line item, where the XML data will conform to a given XML Schema (provided by a reference) where the elements, attributes, namespaces and values of the XML Schema are grounded in a given formal semantic, e.g., a target ontology.

The problem in outline form:

- Given:
  - Source
    - A relational tuple
    - A relation schema that the tuple conforms to
    - A relational domain grounded in a formal semantic for each relational attribute in the relation schema
  - Target
    - An XML Schema
    - A formal semantic grounding for each element, attribute, namespace and value of the XML Schema.
- Derive:
  - A transformation of the source tuple into XML structured data that conforms to
  - the target XML Schema and conserves the source semantics.

The approach we describe here begins with the relational tuple and recurses through the sub-problems that must be solved to address the challenge. We

hope to show, by the time we have recursed to the most foundational sub-problem(s), how the overall challenge may be successfully addressed.

Before we launch into the details we preview the process in outline form.

1) Discover pre-existing concrete transformation code and enact it.
2) In the absence of pre-existing code, discover a pre-existing transformation model: interpret the model or compile the model to code and enact it as in step (1). Persist generated executables for future reuse.
3) In the absence of a pre-existing transformation model, discover a pre-existing language mapping and ontology alignment mapping, reason the way to a transformation model from these mappings. Continue with the remainder of step (2).
4) In the absence of a language mapping or an ontology alignment mapping, discover the formal, compositional definitions of the source and target languages and ontologies and derive the language and/or ontology alignment mapping(s) from the definitions. Continue with remainder of step (3).
5) In the absence of formal language or ontology definitions create these as needed. Continue with remainder of step (4).

Concrete transformation code

As the relation schema that our tuple conforms to is known by a unique identifier within the overall system, and since the same is true for the target XML Schema, we may use a search and discovery process to determine if a concrete transformation between these two artifacts is already present somewhere in the system, available either locally or remotely. By concrete transformation we mean executable or interpretable transformation code. A search or discovery process may be a local query, a remote query (client/server) or a peer-to-peer discovery mechanism (e.g., JXTA, Groove, Kazaa, Grokster, etc.). Should such transformation code be found then all that must be done is to retrieve and enact it locally or alternatively, enact it as a remote service.

In the event we do not discover a pre-existing solution we take the next step: we look to discover, through a discovery process of the kind just described, an existing transformation model – a mapping - between the source and target schemas. We will now describe and illustrate an example mapping.

Transformation Models

> *Note to the reader: This section presumes an understanding of the metamodel hierarchy, from instance data through models, metamodels and MOF, the Meta Object Facility serving as the meta metamodel, plus the interrelationships between these. An introduction to these concepts is provided in Appendices 1 & 2.*

We begin, in effect, by jumping to the end of this section's story, illustrating what we mean by a transformation model, i.e., a mapping between models expressed as a model that conforms to a transformation metamodel. We will make a few remarks and then develop and explain such mappings from metamodeling first principles.

In Figure 1 *Abstract Syntax Tree for a Transformation Map* we instantiate the Common Warehouse Metamodel (CWM) Transformation metamodel in order to map a relational purchase order line item as source to an XML purchase line item as target.

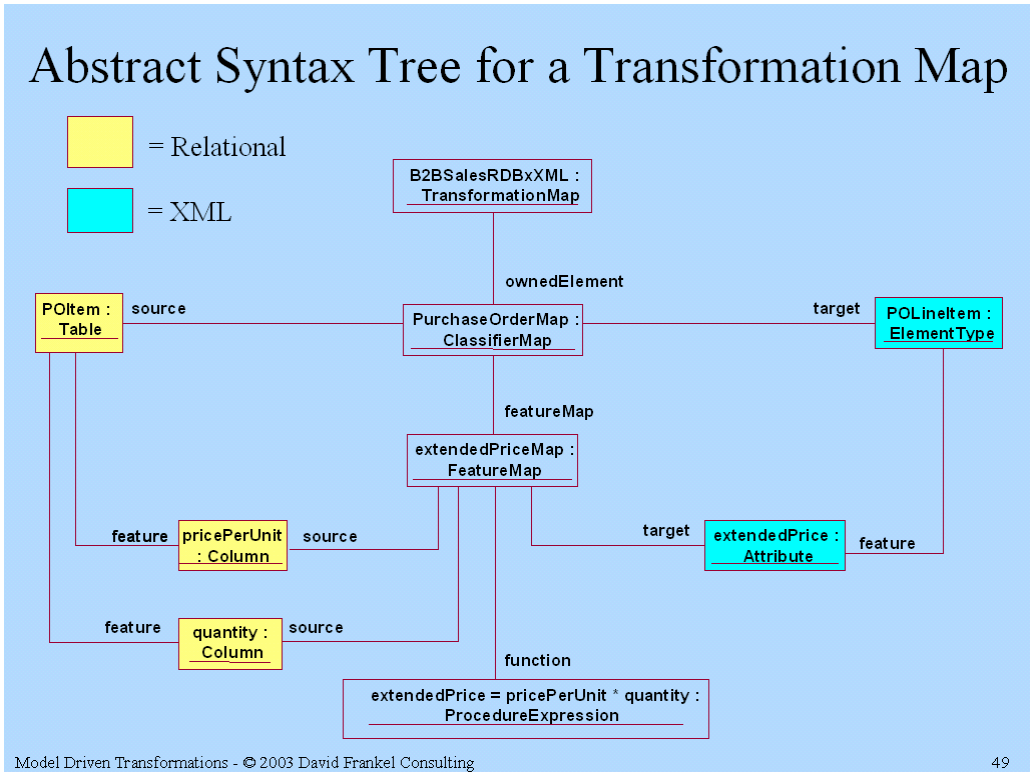


Figure 1 *Abstract Syntax Tree for a Transformation Map*

This transformation, while articulated at two levels (the metamodel and model levels) has consequence on three layers. At the metamodel layer it defines that the Classifier "relational table" (i.e., relational entity type), will be transformed to become the Classifier "XML elementType" and that the Feature "relational column" will be transformed to become the Feature "XML attribute".

At the model layer it defines that the relational table "POItem" will be transformed to become the XML element type "POLineItem" and that the relational columns "pricePerUnit" and "quantity" will be transformed according to an expression to become a single XML attribute, "extendedPrice".

Finally, at the instance layer, it has the consequence that, for each row of relation POItem, the values of pricePerUnit and quantity must be multiplied to produce the value of the XML attribute extendedPrice of the XML element that represents the transformed instance.
We note in passing that the label "procedureExpression" of Figure 1 is somewhat misleading: it need not be a procedural expression; it may also be declarative.

When we referred in the last section to "discovering a mapping" this is the kind of thing we meant to discover. A mapping, as we are using the term here, is a model, and models are not just pictures. The graphical view of a model is just that, a specific type of view. A key aspect of modeling is the ability to view and interact with models in many ways – as graphics, as structured documents, through an API, even as binary – all of them equivalent semantically. This applies to mapping models as well as to other kinds of models. Thus, we may discover a mapping persisted as a document stored in a repository. Alternatively, we may interface programmatically with a MOF repository using an API such as Java Metadata Interface (JMI) to access and navigate a mapping that is managed by the repository.

With some appreciation of what a mapping model is we may now return to basics and explain where mappings come from and how they work.

We begin by considering a family of metamodels called Common Warehouse Metamodel, or CWM[1] for short. It probably should have been called Common Warehouse Metamodels, noting the plural, for that is what it offers.[2]

CWM consists of a family of metamodels defined by meta-metamodel constructs that are, for the present purposes, practically equivalent to those of MOF. The purpose of CWM is to enable information flow across a range of logical and physical data models. The family includes metamodels for relational, multidimensional star-schema, record-structured, XML, hierarchical, and other data models. What is unique, and what makes this a family and not just a collection, is that all of the CWM metamodels conform to a common meta-metamodel, essentially MOF.

[1] Common Warehouse Metamodel is a standard of the Object Management Group.

[2] For a more complete discussion of these concepts see David Frankel's book, Model Driven Architecture.

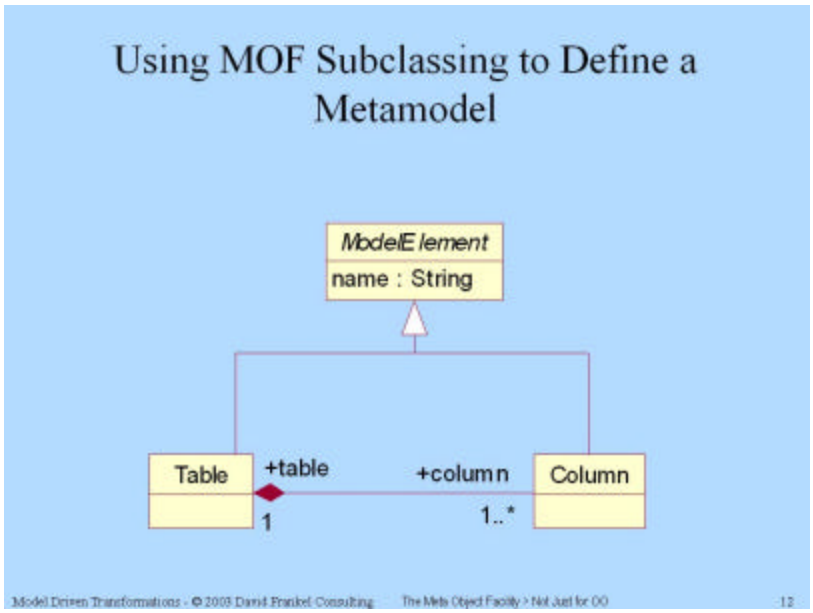


Figure 2 *MOF Subclassing*

In Figure 2 *MOF Subclassing* we show how a very simple, tabular data structure can be defined as a MOF conformant metamodel, including a constraint that a table owns its column set.

Now let's take a look at the CWM metamodels for relational and XML, the subject of our running example.

Figure 3 *CWM Relational DB Metamodel* illustrates how meta-metamodel constructs from MOF, namely Classifier and Feature, are instantiated at the metamodel layer (remember, MOF and its constructs are at the meta-metamodel layer) as CWM Common metamodel constructs and then specialized so as to define relational constructs, such as table and column.

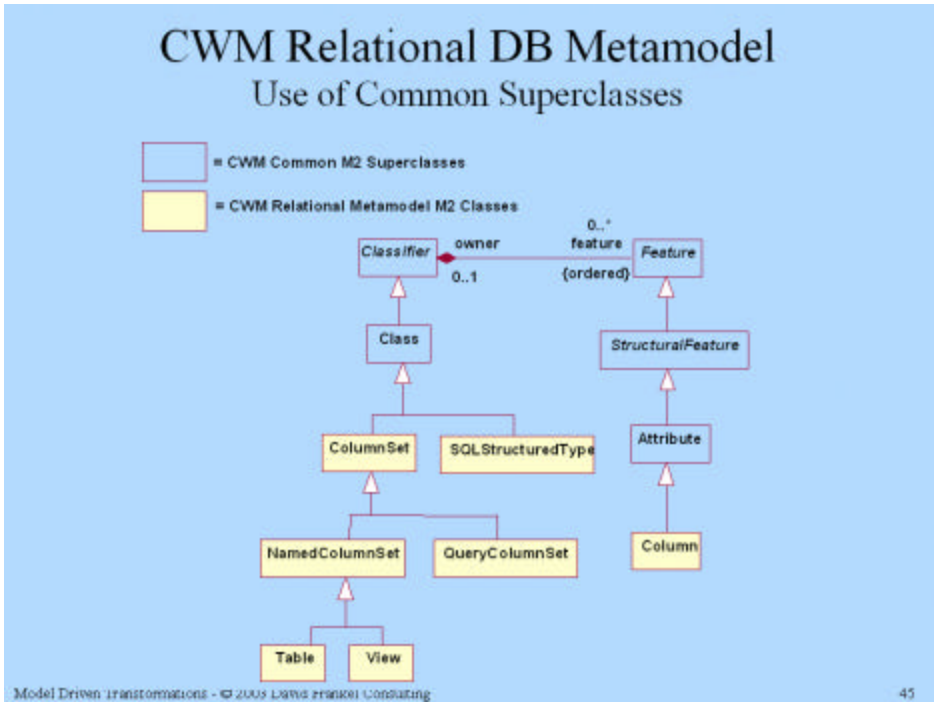


Figure 3 *CWM Relational DB Metamodel*

We can impose further constraints in the metamodel, as is shown in Figure 4, depicting a fragment of the CWM relational data metamodel.

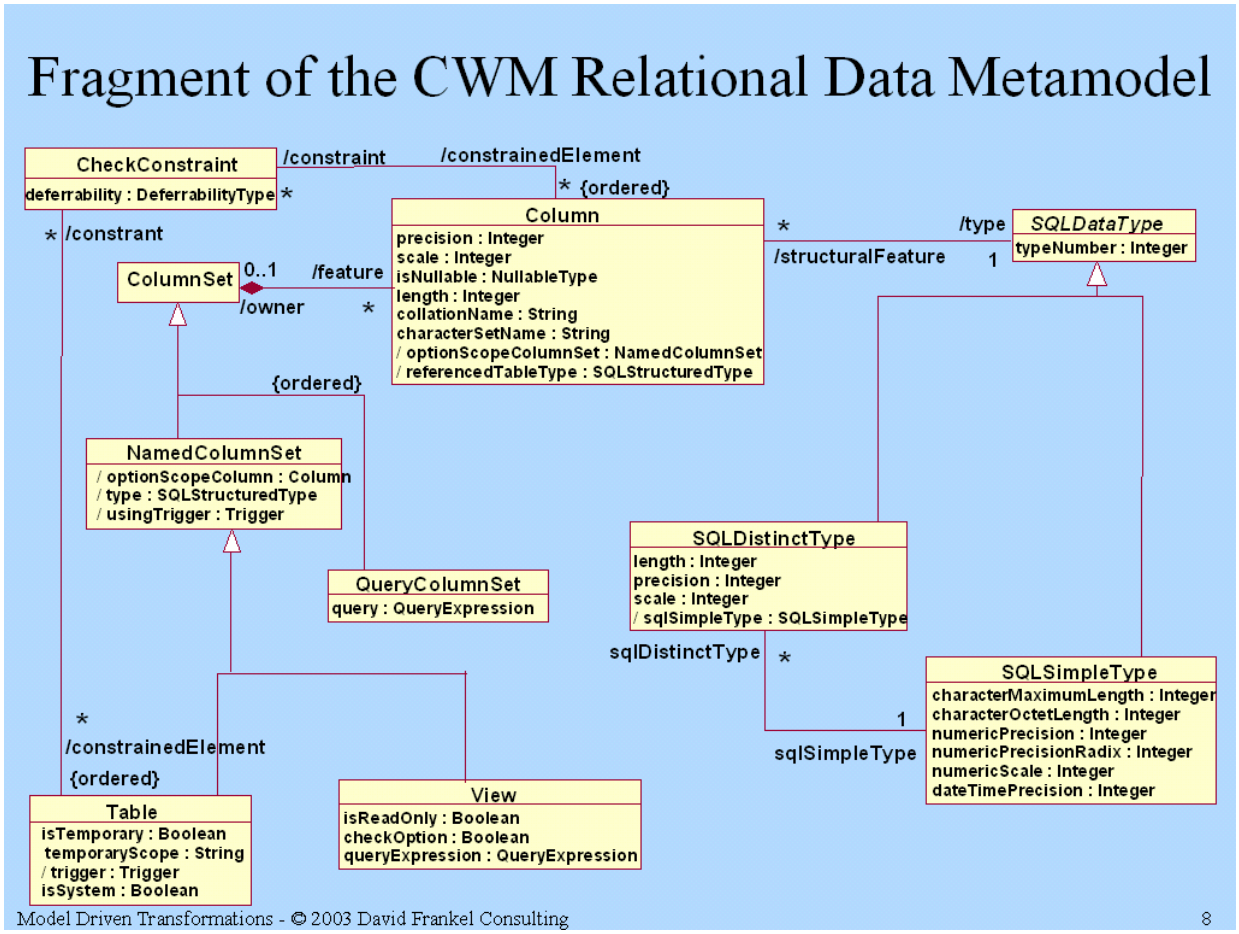


Figure 4 ***Fragment of CWM Relational Data Metamodel***

We can also illustrate a sample relational schema that conforms to the CWM relational metamodel, as in Figure 5. Here we depict the schema as an instance of the metamodel.

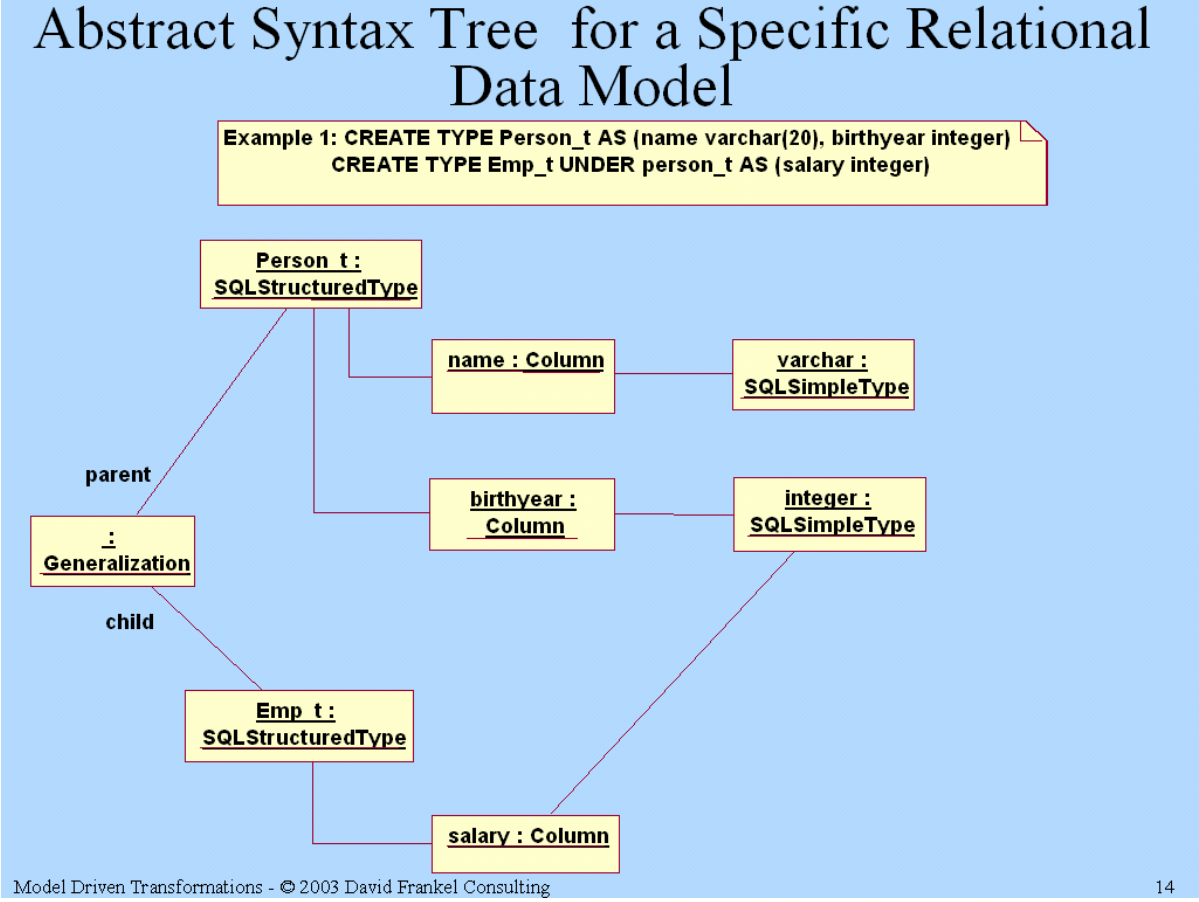


Figure 5 ***Specific Relational Data Model Instance***

Next we examine how meta-metamodel constructs, again Classifier and Feature, are instantiated by CWM at the metamodel layer and specialized to define XML constructs, namely element type and attribute, as is shown in Figure 6.

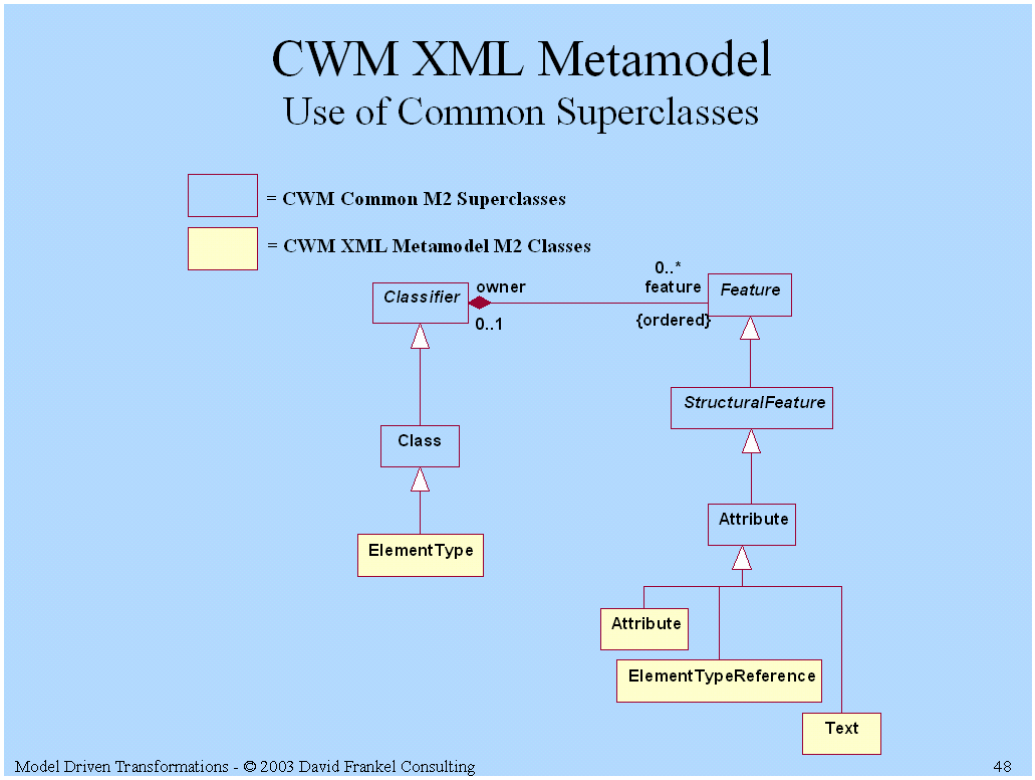


Figure 6 *CWM XML Metamodel*

Notice how both the relational and XML metamodels are defined in CWM by the same constructs, ultimately instances of MOF Classifier and Feature.

While most of the data models that are now supported by CWM (as metamodels) were *developed independently* of the others, because their CWM metamodels conform to a common meta-metamodel they may be interrelated, specifically mapped to one another. This is a key concept and we examine it in the next section.

Mapping source to target metamodels

A transformation model requires both a metamodel mapping and a model mapping that is constrained by the metamodel mapping. The groundwork for a metamodel mapping has now been laid, as illustrated in Figure 7 *Common Core for UML, CWM, and MOF*.

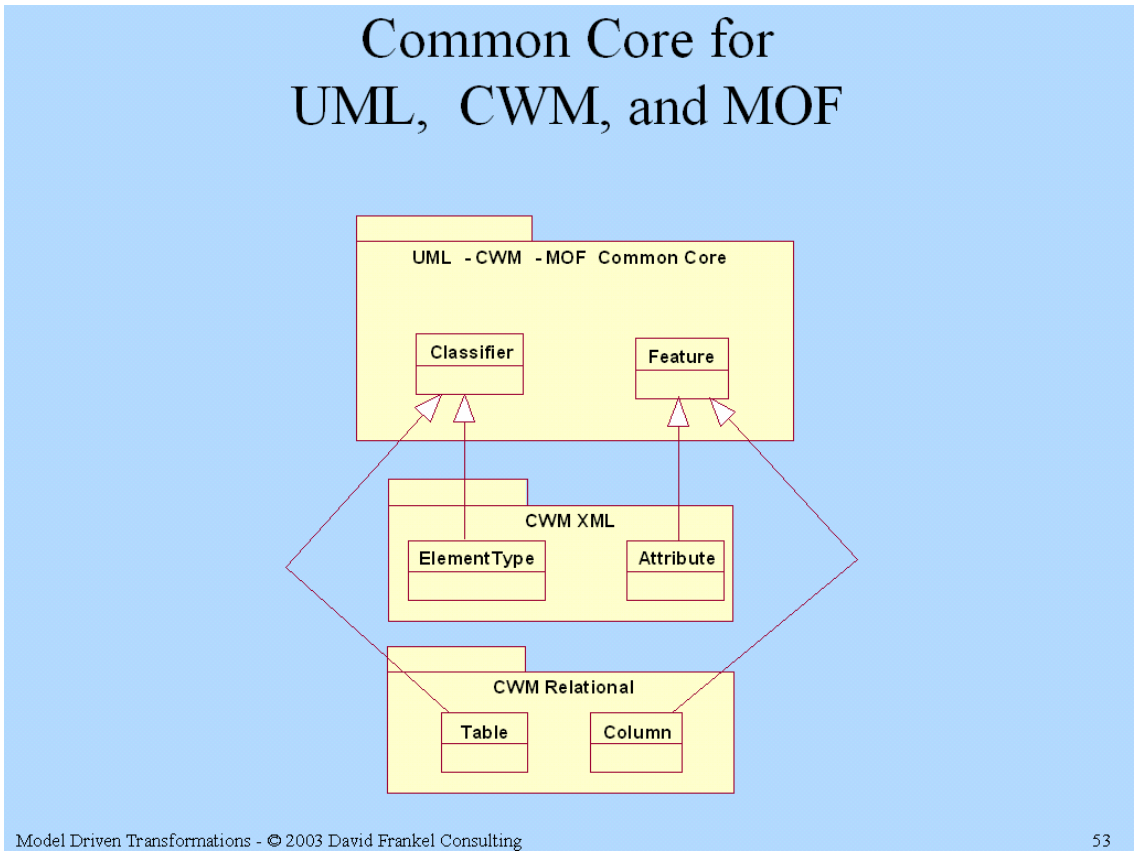


Figure 7 *Common Core for UML, CWM, and MOF*

Elements of metamodels representing distinct systems may be defined using common and relatable constructs. For example, the notion of a relational column and an XML attribute are structurally quite distinct and governed by different constraints, yet we have defined them using an identical metamodel construct, Feature.

We now look at the means to map common metamodels to one another. Figure 8 illustrates a fragment of the CWM Transformation metamodel that enables this.

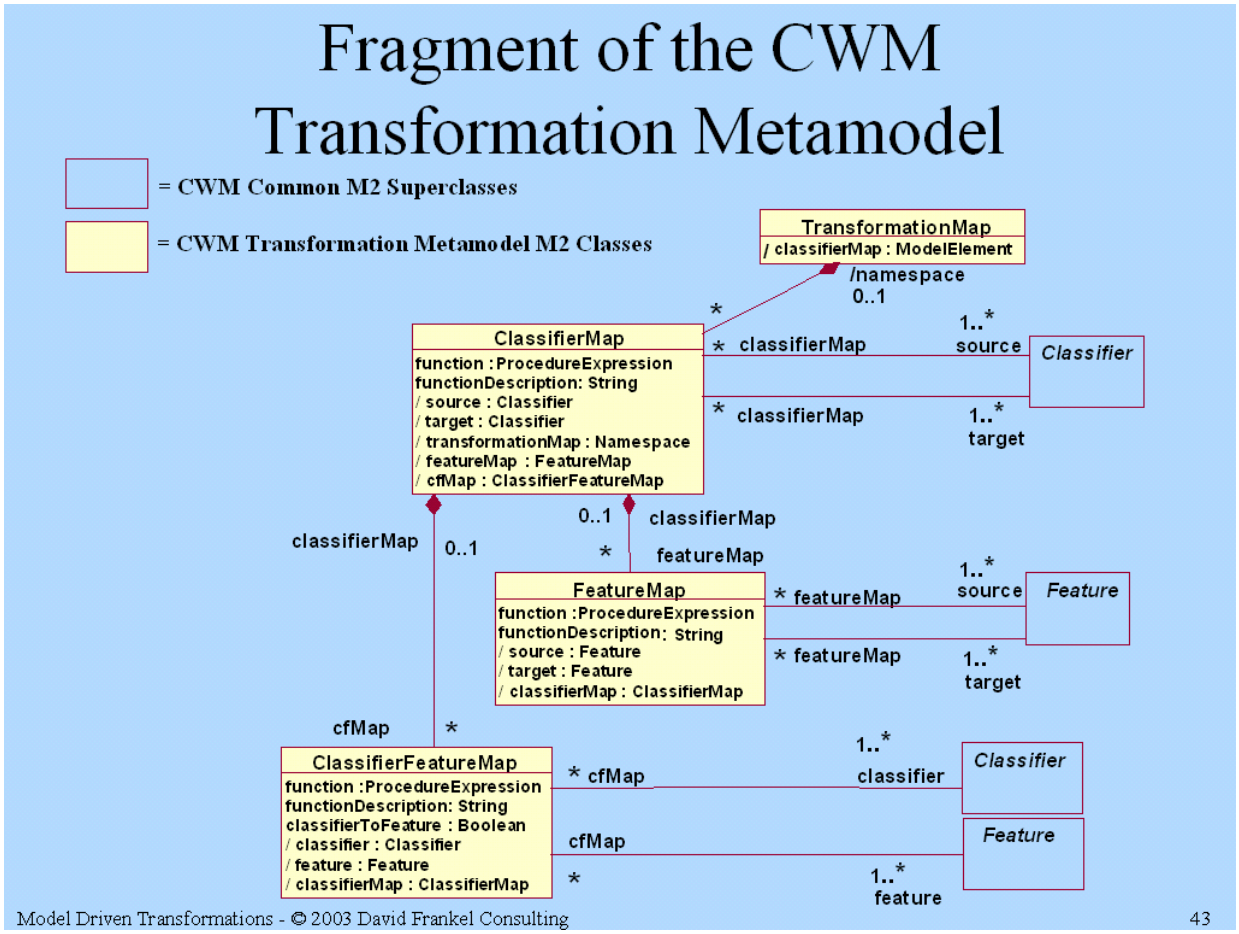


Figure 8 ***Fragment of the CWM Transformation Metamodel***

Note how the transformation metamodel enables Classifiers to be mapped to Classifiers; Features to be mapped to Features; and Classifiers to be mapped to Features. This becomes clearer if we make things more concrete by returning to the specific mapping of a relational purchase order item to an XML purchase order line item.

In Figure 9 *Abstract Syntax Tree for a Transformation Map*, we instantiate the CWM Transformation metamodel in order to map a relational purchase order line item as source to an XML purchase line item as target.

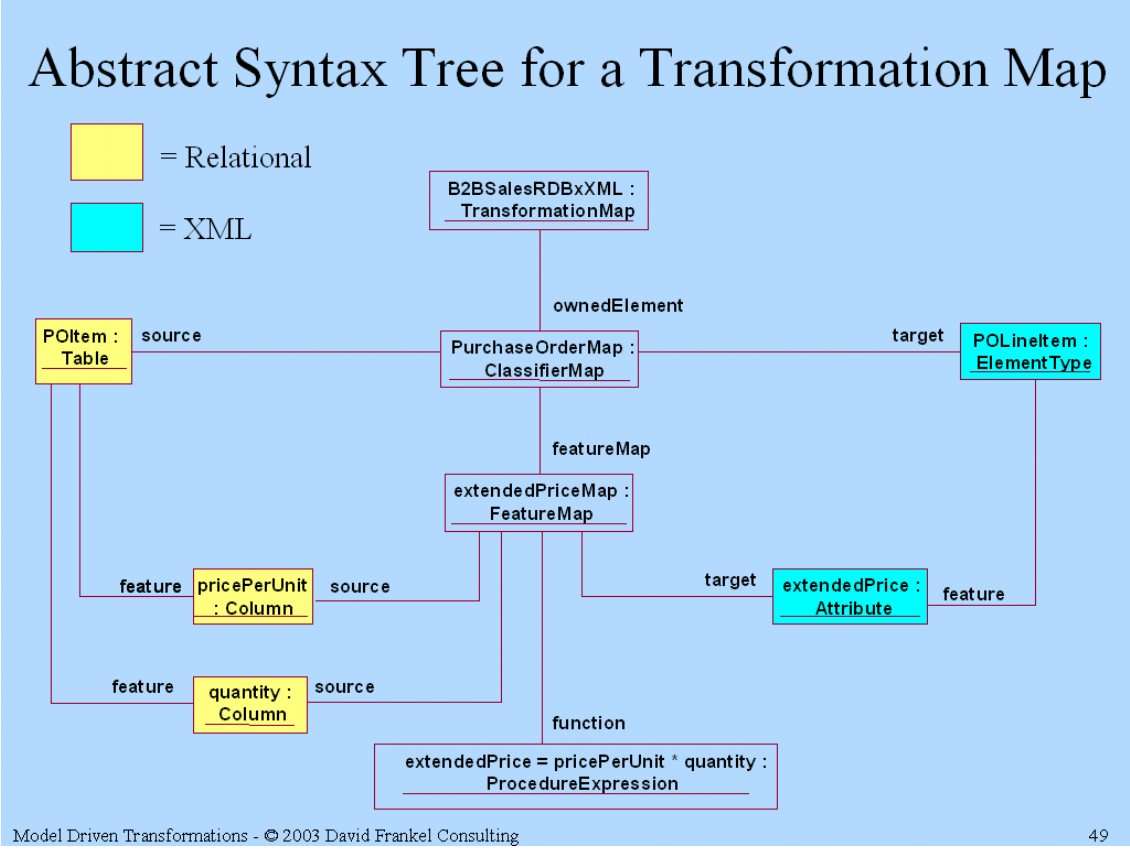


Figure 9 *Abstract Syntax Tree for a Transformation Map*

The mapping explicitly defines not only what relational metamodel constructs are mapped to which XML metamodel constructs (e.g., relational table to XML element type) but also which specific entities of the source schema are mapped to specific entities of the target (e.g., POItem maps to POLineItem). Both structure and semantics are mapped by this one mapping model. It may be noted in passing that the function responsible for the mapping may be used to translate between systems of units or value or such may be formalized as part of the mapping itself (the details are beyond the scope of this paper).

CWM, and a metamodeling approach to transformation in general, is not limited to relational and XML however. Figure 10 provides just a sampling of the many source/target transformation pairs that CWM facilitates via metamodels and mappings.

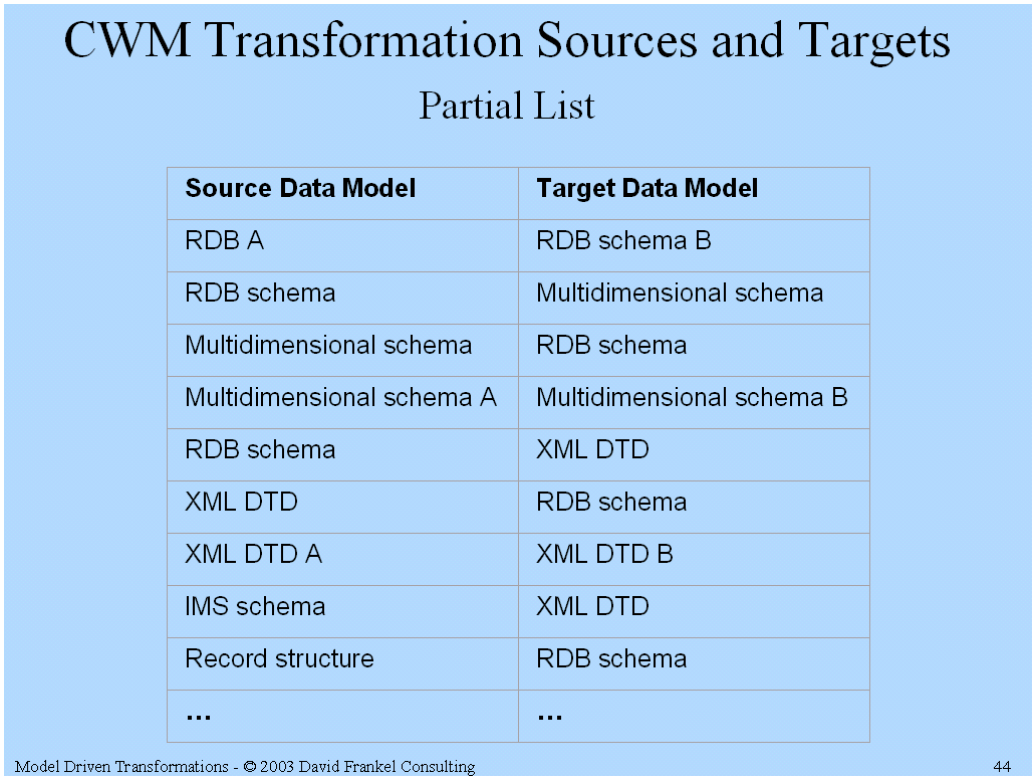


Figure 10 *CWM Transformation Sources and Targets*

The transformation possibilities offered by CWM are extensive, and cover most needs of the data warehousing community, to the extent that the major vendors in the space, IBM and Oracle, have embraced CWM and offer implementations of the standard. CWM however, provides a specific collection of metamodels out-of-the-box, and this collection does not provide for all possible transformation needs, which is more extensive than moving instance data between Online Transaction Processing Systems and data warehouses.

Since late 2002/early 2003 the OMG has been working on generalizing CWM to support all types and varieties of MOF-based transformations This idea, of CWM-like transformations built directly around MOF 2.0, is supported by a current standardization effort at the OMG that is already well along, defining MOF-based transformations as part of the MOF 2.0 QVT specification (Query/View/Transformation). QVT will support all of the mappings we've shown here plus much more.

Should our discovery process find a pre-existing mapping we can use Model Driven Architecture (MDA) code generation technology to generate executable

transformation code. Alternatively, we may use a model interpreter to interpret the transformation mapping model at runtime to produce the target XML representation. Generated transformation code may be persisted as an artifact with an association created between the source and target schemas and the newly generated transformation code, creating an asset that may be discovered and reused in the future.[3]

Deriving transformation mappings

In the event we do not discover a pre-existing transformation mapping model we set out to derive one from things more foundational. Things more foundational include a transformation model that maps the source language, i.e., the relational model of data, to the target language model, i.e., XML Schema conformant XML. We call this a language mapping.

Language Mappings

A valid language mapping was implicit to the relational to XML line item mapping we just explored in Figure 9. We now make the language map, at the moment a very simplistic one, explicit.

Figure 11 and Figure 12 illustrate two equally valid ways of mapping the structure of relational data into the structure of XML data. In the first we map relational table to XML Element type (the same mapping that was implicit in Figure 9). In the second we show a distinct – but equally valid – mapping, where a relational attribute is mapped to an XML Element type, but in this case an element that is the child of the XML Element that represents the relational table.

[3] In other reports we have referred to the functionality discussed thus far as MRT – Model-driven Representation and Transformation.

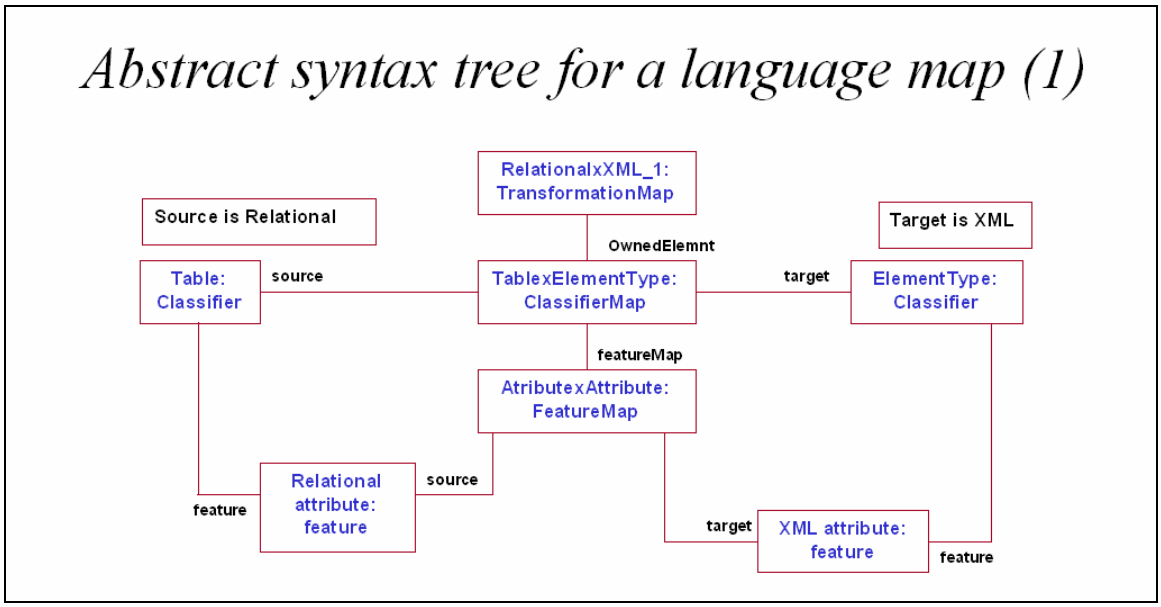


Figure 11 ***Abstract syntax for a language map (1)***

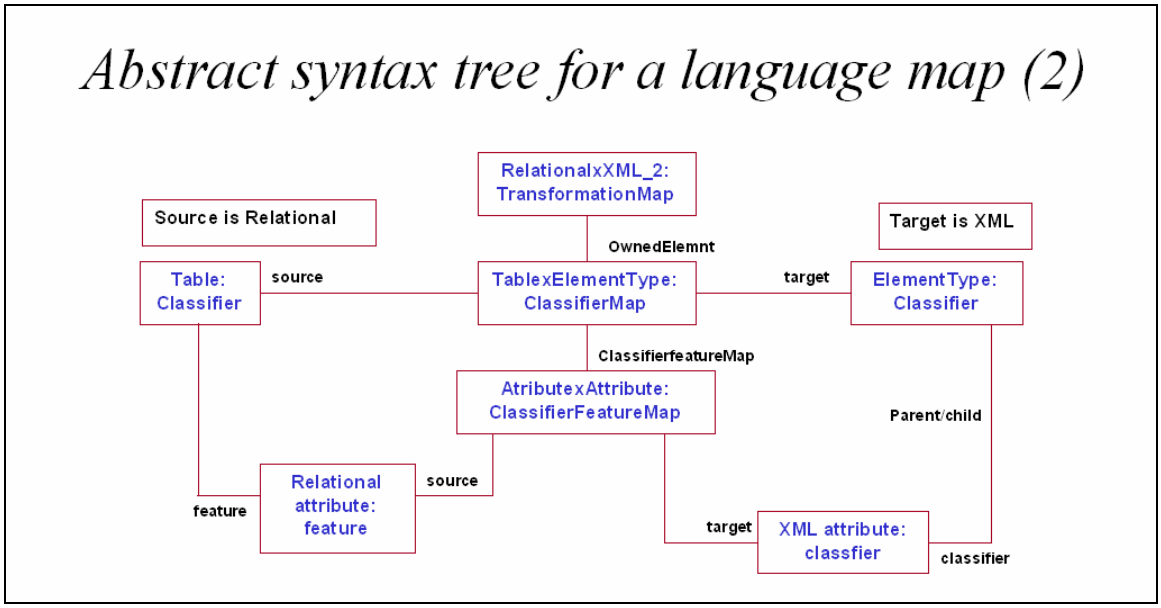


Figure 12 ***Abstract syntax tree for a language map (2)***

This diversity of mappings is a consequence of the many ways in which languages, XML in this case, may be employed in a representational task. After all, it is the developer's choice whether to represent a data item as an attribute of an XML element or as a child element. The multiple possible mappings support the many ways in which a language may be validly employed. We refer to this expressive range, and the choice of a style that selects one from the many, as *representational idiom*. Melnik and Decker, writing in a paper on the semantic web, illustrated six distinct means of using Resource Description Framework (RDF) to represent the statement that "Mozart was the principle composer of the

Requiem but with assistance from Salieri" (ignoring for our purposes that this belief is a myth not supported by historical fact).

While metamodel based techniques can express the many transformations between languages they do not in and of themselves dictate "the right one" to be used in each specific situation. However, where a representation model, source or target, unambiguously designates the representational idiom at work, then it is possible to automatically select a corresponding transformation mapping.

In the B2B example both source and target schemas could reference the representational idiom(s) at work (e.g., employing an idiom on the target side that uses XML attributes rather than child elements), eliminating the ambiguity in selecting a language mapping.

Put simply it comes down to this: a language is not completely specified unless and until it is coupled with an idiom (moreover it can be argued that even with a designated idiom a language specification is incomplete, requiring the further designation of an ontology). When source and target languages are identified, complete with source and target idioms, then a language transformation may be selected (or derived/constructed in its absence).

Ontology Alignment Mappings

A language mapping model is not the only thing more foundational that we require for our derivation of a transformation mapping model from things more foundational. Besides the identification of idioms involved in a language mapping we also need a designation of how meaning embodied in the source representation should be mapped to meaning as it is embodied in the target representation. At minimum this requires a transformation that maps the formal semantics of the source schema and data into the formal semantics of the target schema and data.

The semantics that govern the relational line item source in the B2B example is provided by a community-wide semantic called Open Buying on Internet (OBI) while the Universal Business Language (UBL) provides the semantics of the target. These two standards represent the agreed upon semantics of their community respective communities, though neither of these is a full-fledged ontology (UBL is heading in that direction).

For the purpose of our example we can treat both OBI and UBL as ontologies. For reasons of manageability, interoperability and productivity, we choose to author, view, manage and manipulate ontologies as models. Specifically, ontologies may be represented as models that conform to a MOF-based Ontology Definition Metamodel (ODM). The transformation model we need to transform from the source ontology to the target ontology is therefore an ontology alignment mapping.

We now take a brief look at the ODM. Note that the ODM is a work in progress at the OMG, slated for adoption in late 2004 or early 2005. The snapshot we'll be reviewing is from late 2003, with several revisions having been made since that time. After we present the snapshot we'll highlight the major changes.

ODM is a MOF conformant metamodel intended to support models with the expressive power of RDF(S), DAML+OIL, OWL (all levels), KIF, Conceptual Graphs, and Common Logic. In other words, the metamodel supports the concepts, structures and constraints of these languages with no loss of fidelity. See Figure 13 *Layered Ontology Definition Metamodel (ODM) Approach*.

ODM will include standardized mappings to several knowledge representation languages, e.g., IBM research, a member of the Ontology Definition Metamodel submission team, is developing RDF(S) and OWL mappings to ODM's core. Sandpiper Software is a key player in the effort; also supporting the effort are AT&T/Gentleware and DSTC.

Models that conform to ODM may be authored in IBM Rational Rose® supplemented with a plug-in that supports the associated UML profile, such as is provided by Sandpiper's Medius® Visual Ontology Modeler (VOM). Representations in the supported languages may then be imported and exported based on the standardized mappings. Models that conform to ODM may exploit XMI for tool and repository interchange and MOF repositories for management.

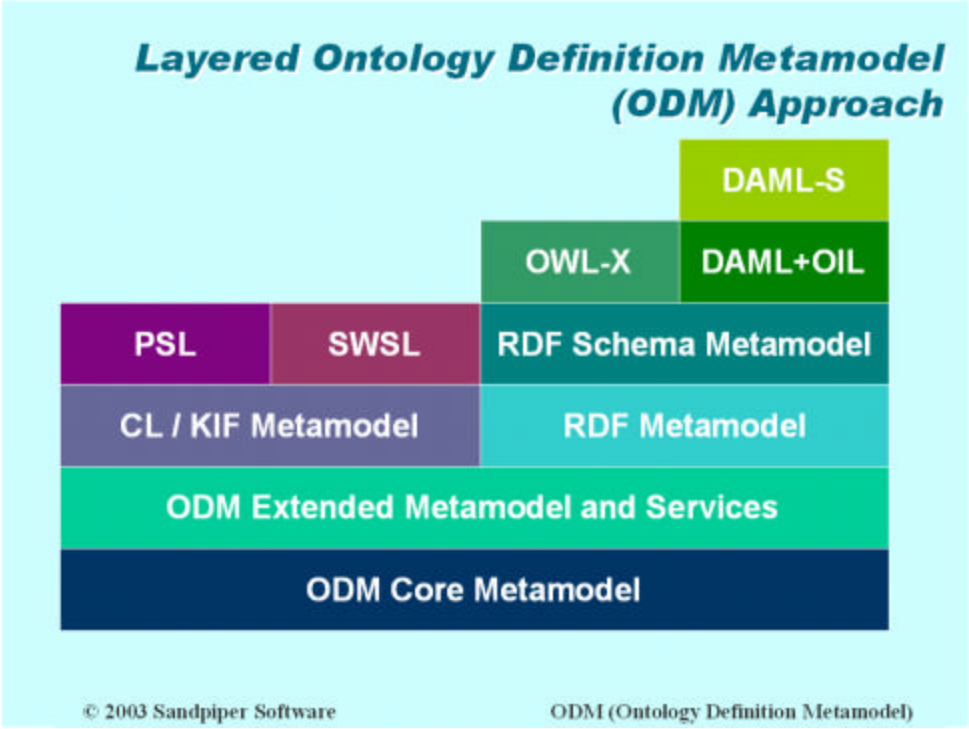


Figure 13 ***Layered Ontology Definition Metamodel (ODM) Approach***

The core metamodel will be an abstracted representation that leverages interoperability notions derived from frame-based systems such as OKBC (Open Knowledge Base Connectivity), though it will not depend on OKBC, or frames, in fact. It will be supported by a model-theoretic semantics.

A central notion in the development of the core metamodel is that "everything is a relation". Classes and individuals are unary relations, slots and RDF properties are binary relations, facets are ternary relations, etc.

The core of ODM, supported by the Sandpiper plug-in to IBM Rational Rose, has been in use for some time as part of HORUS, a DARPA program utilizing ontologies for intelligence community needs.

> "The focus of Horus is to enable and exploit semantic-based markup of sources to promote information discovery and integration, ultimately by software agents as well as humans. Users and agents will access, manipulate, and create knowledge that is organized as Horus "knowledge

> objects". These (conceptual) objects represent real-world entities such as military units, terrorist organizations, and geopolitical events. Information in knowledge objects is linked to its source (i.e., a database or web page). This supports the maintenance of information pedigrees and drilldown to the original sources. User sites will build portals to provide access to these objects, resident in a Horus Knowledge Base (KB)."[4]

Regarding tools used in the Horus project, ISX states:

> "Authoring tools enable a (trained) user to define classes and properties and specify their interrelationships via graphical user interfaces. These tools output ontologies in an OBML [*OBML refers to Ontology Based Markup Language, e.g., DAML. Note added for clarity – not part of cited source.]*. Commercial companies, the RDF community, and the DAML project have built a number of tools for authoring and validating ontologies (and schemas). We have used COTS tools such as XML Spy™ (www.xmlspy.com) and Sandpiper's Visual Ontology Modeler (www.sandsoft.com)." [5]

An overview of ODM, seen in Sandpiper Software's Medius Visual Ontology Modeler (VOM) is shown in Figure 14. A wider view of the same ODM overview is depicted in Figure 15.

[4] Brian Kettler, ISX Corporation

[5] http://semanticobjectweb.isx.com/isx-sow-wp-2002-03.pdf

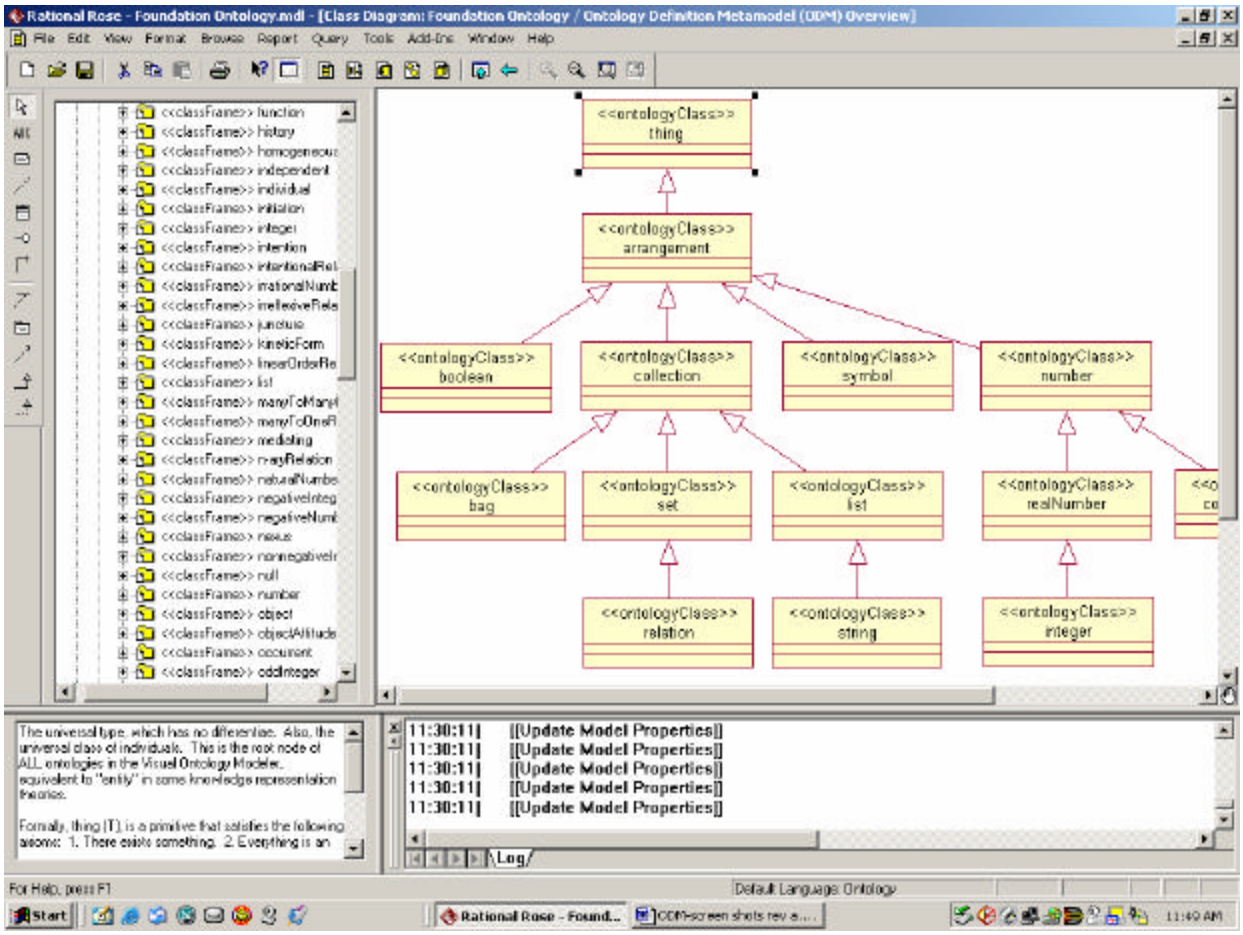


Figure 14 *Ontology Definition Metamodel in Visual Ontology Modeler (VOM)*

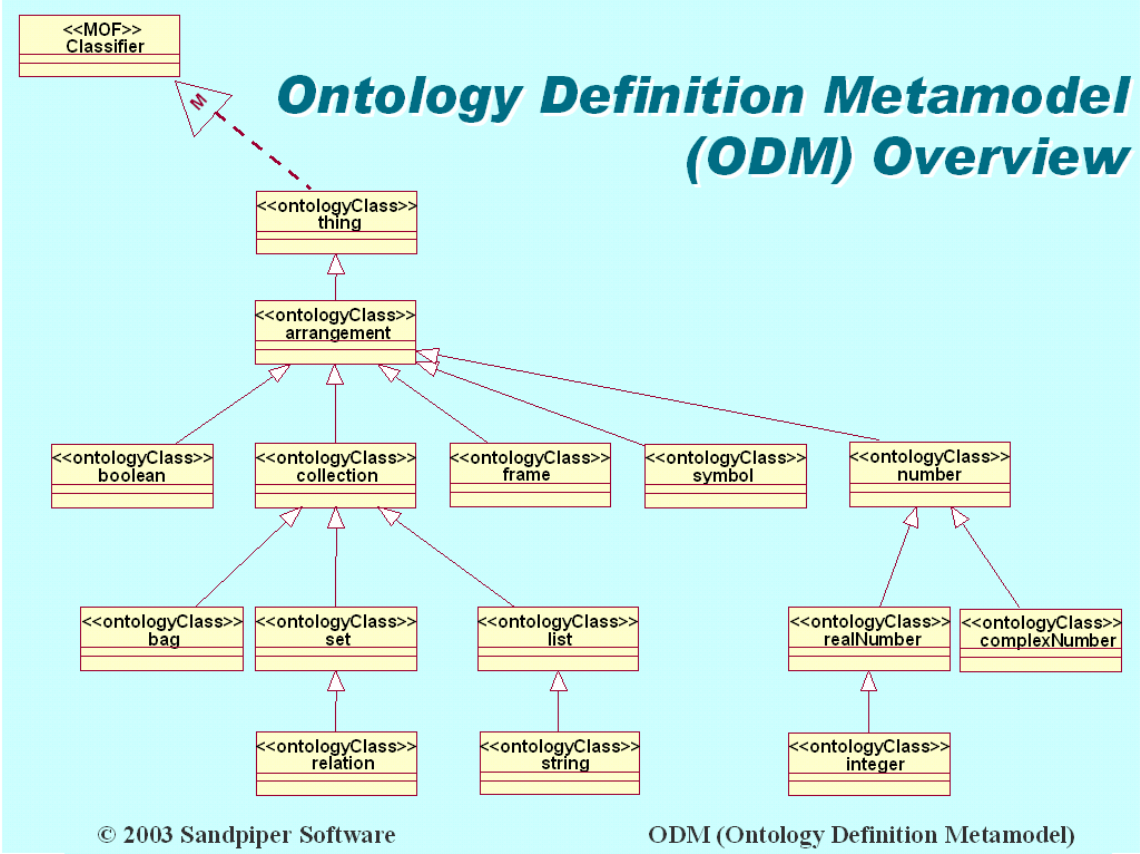


Figure 15 *Ontology Definition Metamodel (ODM) Overview*

Some important features of the ODM:

- All ODM Core elements are relations (see Figure 16 *Ontology Definition Metamodel (ODM) Core (initial)*)
  - Preserves semantics of individuals and classes as unary relations.
  - Relation is a MOF Classifier with multiple inheritance support, thus all core elements including individuals and facets can support inheritance; constraints on inheritance are KR language specific and therefore provided in metamodels layered on ODM Core.
  - ODM Core supports n-ary relations; relations do not require defined endpoints (critical in some knowledge representation languages)
  - Supports incomplete definitions and partial knowledge, including incomplete specification of individuals (common in KR)
- Use of packages to emulate frames
  - Provides a consistent mechanism for keeping details together (slots, facets, axioms, diagrams, deployment details).
- Ontologies and frames are managed as separate components
  - Facilitates collaborative, component-based development, reuse, configuration management, ontology restructuring and maintenance.

The Sandpiper specific implementation of the ODM, in its Medius Visual Ontology Modeler (VOM), supports these by emulating key features of some frame systems in addition to supporting XML-based description logics languages.

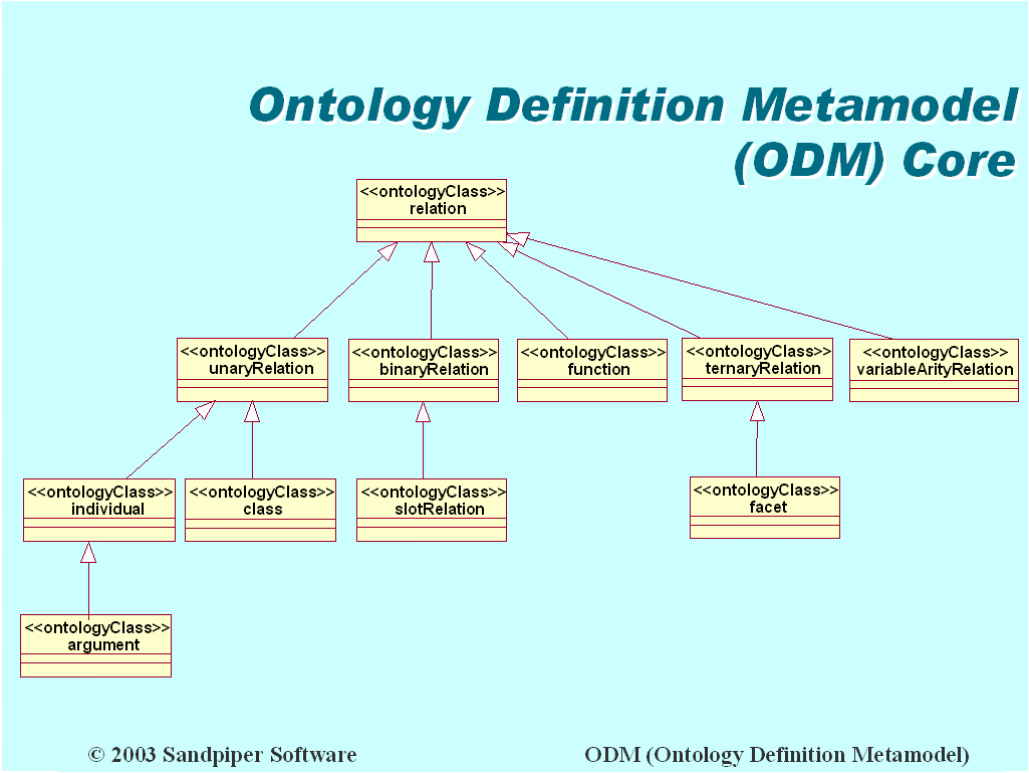


Figure 16 ***Ontology Definition Metamodel (ODM) Core (initial)***

The ODM has seen two major changes since the snapshot seen in Figure 16. The ODM as presented here borders on being an ontology definition meta-metamodel – a model for defining ontology definition metamodels. Such would require a major rework of MOF itself. This motivated the first change, to reconceive ODM as a collection of metamodels to suit the range of ontology languages and representations in them, in the same spirit as CWM but for ontology languages rather than data languages. Second, Common Logic (CL) was itself evolving into Simplified Common Logic (SCL), and ODM was revised to reflect this change. Overall, the revised ODM presents two packages, one for SCL and ontology languages that may be represented by it (e.g., KIF, FOL, Conceptual Graphs), and one for OWL, with the two packages extending a common core.

Let's return to our B2B example and apply the ODM. In this example we ground each model, source and target, in a semantics that is standardized across each one's respective community. RosettaNet's Open Buying on Internet (OBI), though not a formal ontology, does provide an abstract model and semantics for

ecommerce that is shared across its community. OASIS's Universal Business Language (UBL) has gained even wider acceptance as a community-wide shared semantics and, while not currently articulated in an ontology or knowledge representation language, is moving in this direction, supported by ongoing collaboration between OASIS and the W3C's web ontology effort. Applying OBI and UBL in this example conveys the essential concepts while keeping the example clear and simple; use of large formal ontologies in this example, such as Cyc, would have added needless complexity and drawn attention away from the key points.

To achieve the grounding we first express OBI and UBL in terms of the Ontology Definition Metamodel (ODM). Next, each element of the source model becomes a subclass of an OBI element. Similarly, each element of the target model is subclassed to a UBL element. This is shown in Figure 17 and Figure 18.

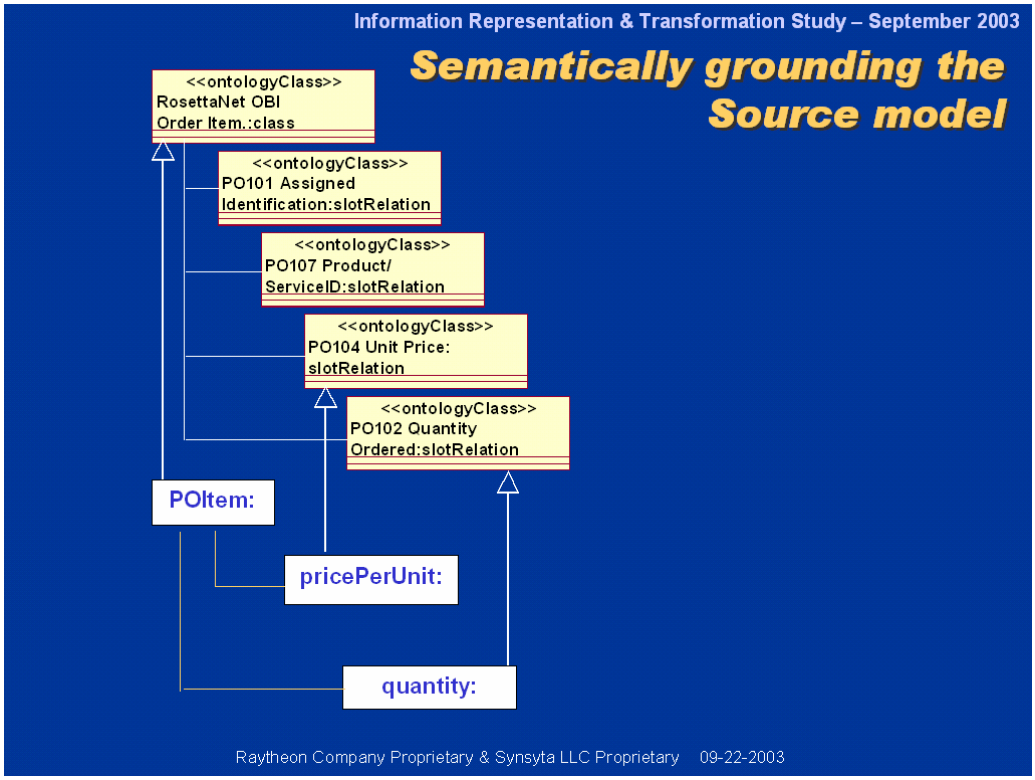


Figure 17 ***Semantically Grounding the Source Model***

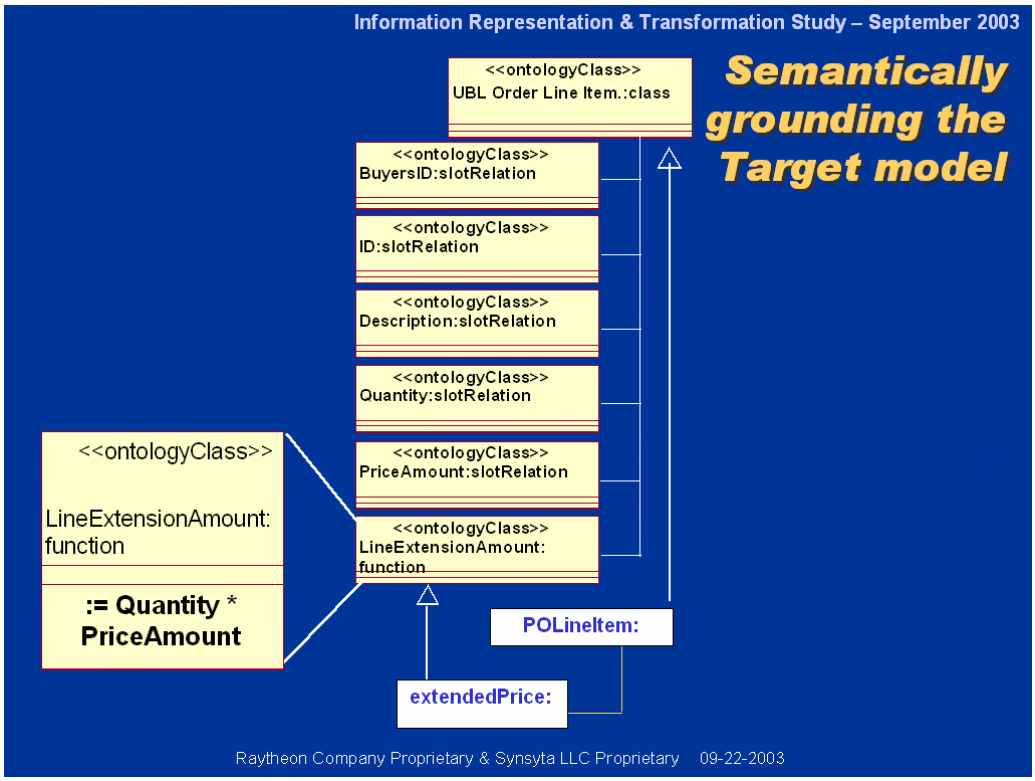


Figure 18 ***Semantically Grounding the Target Model***

Since we have expressed both source and target semantics as models conforming to ODM we may now apply a CWM-like transformation mapping model to align these semantics: we may define a transformation map between the ODM conformant models that represent the semantics of OBI and UBL. This is depicted in Figure 19.

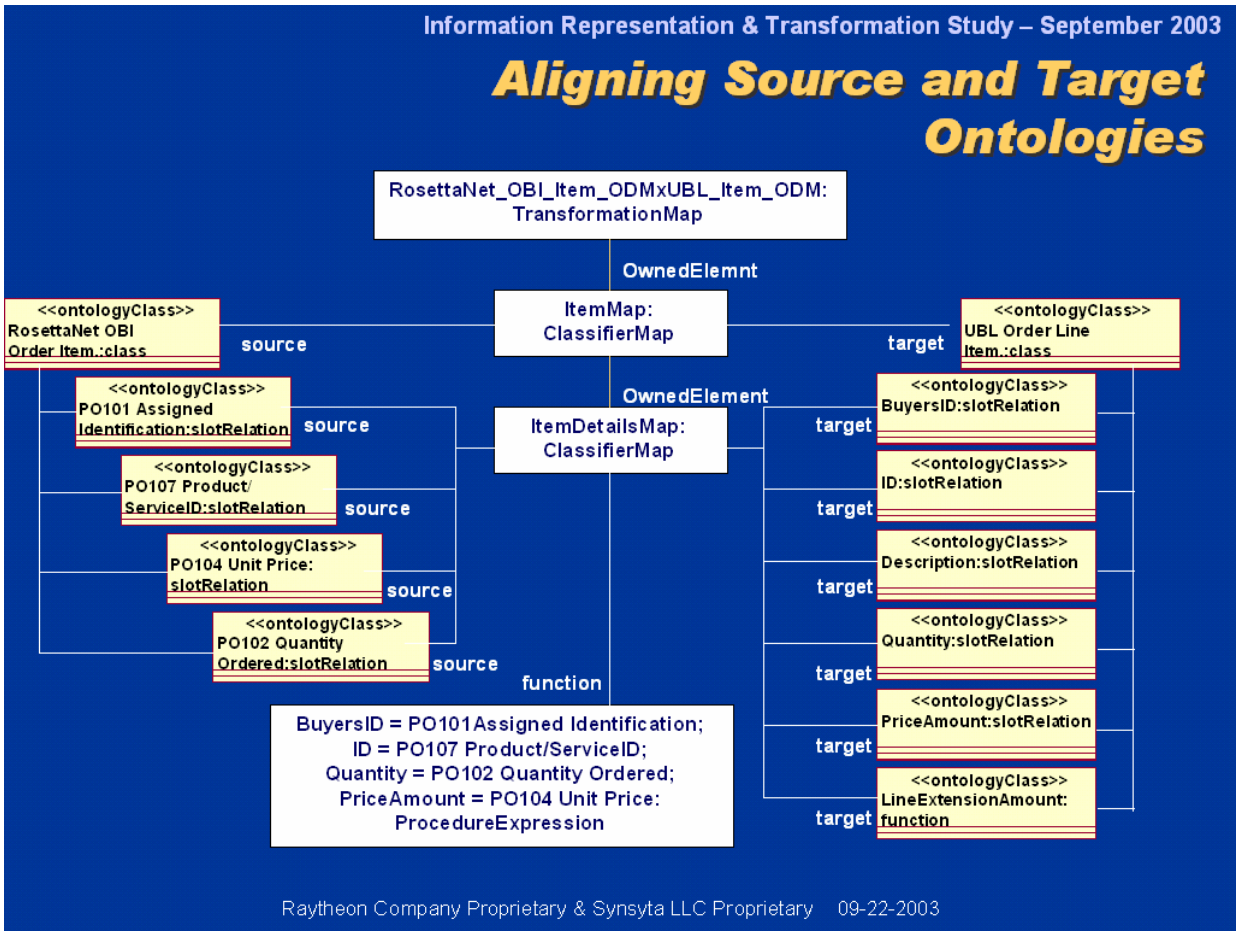


Figure 19 ***Aligning Source and Target Ontologies***

Returning to our example, our current aim is to derive a source to target schema transformation model (a mapping model, as depicted earlier in Figure 9) from the combination of a language mapping and an ontology alignment mapping along with the source and target schemas. In keeping with our strategy we first seek to discover a pre-existing language mapping and ontology alignment mapping.

If we discover both of these mappings an automated reasoning method can determine that POItem is semantically equivalent to POLineItem and create a map between them. While the source has no semantic equivalent to the target's extendedPrice the reasoning method can construct a source expression that is semantically equivalent, the product of the source's pricePerUnit times quantity, and can then map this expression to the target's extendedPrice. A reasoning method can thereby automatically produce a transformation model between the source relational schema and the target XML Schema.

The symbolic reasoning techniques we consider to enable automated identification and/or construction of semantic equivalencies include entailment, deduction, induction, abduction and graph planning. We look to the efforts of

those working on Semantic Web Services (SWS) for techniques and experimental results to guide our efforts in accomplishing this derivation (SWS has been focused on automated composition of semantically described web services using entailment and automated graph planning).

We emphasize that we can use a CWM-like (or MOF QVT) approach to define a mapping between differing semantics. Because of the expressivity of ODM we can create ODM conformant models of the well-known ontologies, such as Cyc, SUO, SUMO and John Sowa's upper ontology; we chose not to here in order to keep the example and illustrations simple. However, this can be done, and that's the point: that we can apply MDA methodology and commercial MDA tooling to ontologies and knowledge representations; that we can use MDA machinery to align ontologies, author ontologies, revise ontologies, manage ontologies, interrogate ontologies, serialize/deserialize ontologies as XML and thereby distribute ontologies, and even generate code from ontologies! And as this simple example illustrates, enabled by MDA methodology and machinery, ontologies may be brought to bear directly within information management and software development environments, methodologies and processes. This is a key leverage point.

It should further be noted that by conducting alignment at the more abstract ontology level, rather than the more concrete model level, several important benefits accrue.

- Greater productivity is achieved as there are far fewer ontologies to be aligned than the models that are grounded in these ontologies, and that the derivation of a transformation map between models grounded in aligned ontologies may be automated.
- Greater consistency is achieved as each ontology alignment cascades to the numerous models that are grounded in such aligned ontologies.
- Greater correctness is achieved as ontologies generally provide richer and more formal semantics on which to base alignment decisions.

Having derived a schema mapping from the more foundational language mappings and ontology mappings we persist it for reuse, associate it with source and target schemas and exit this level of recursion – we may generate code from the mapping or interpret it (as was described earlier).

It is entirely possible that automatic derivation may fail in one of two ways. It may not be able to derive a single schema mapping, but only constrain the solution to one of several. Alternatively, it may derive an incorrect mapping. In the former case, additional contextual knowledge must be brought to bear, such as the objective(s) of the transformation between source and target and the situation the transformation and target representation are a part of (or expected to be a part of). Case Base Reasoning (CBR) may be a useful tool with which to

select the schema mapping that is the best match to the situation. Then too, humans may be consulted, to provide hints, disambiguate or to guide/train the Case Base Reasoner in the preferred trade-offs to make and the balance to be achieved.

The other possible outcome is a transformation failure. Observing a failure presumes a means to monitor usage of the transformed representation, either by humans, applications or both, and determine deviation from expected or desired behavior/performance. When such a discrepancy can be identified then Model Based Diagnosis may be employed to identify the root cause(s) and attempt a repair to the schema mapping. Again, humans may be consulted to aid or accomplish the diagnosis and/or repair. Model Based Diagnosis equipped with a supervised learning mode could profit from such human intervention.[6]

What is to be done if we do not discover pre-existing language and ontology alignment mappings? This is addressed in the next section.

Deriving language and ontology alignment mappings from formal compositional definitions of languages, logics and ontologies

Recursing a level deeper still we now come to the most significant and foundational sub-problems yet encountered here. In the absence of discovering pre-existing source to target language and ontology alignment mappings we aim to derive one or both of these, as required, from things still more foundational. To enable this we now explore a compositional account of languages, logics and ontologies, whereby such artifacts may be formally defined in terms of compositions of fundamental components of language, logic and ontology. Language and ontology alignment mappings are then to be formally derived from the compositional definitions of source and target languages and ontologies.

At this point in our discussion we abandon our B2B example and instead point to three separate but related efforts to provide the required mathematical machinery as well as a basis of confidence that the goals of this section can indeed be achieved. The reason we abandon the example is simple: we have not yet worked out the B2B example directly with the approaches we are about to discuss. We have considered, however, how the methods and techniques of the three approaches could be applied to the challenge at hand. This consideration is, for the time being, abstract.

To achieve the objectives of this section we require a number of capabilities.

[6] The functionality discussed, inferring model mappings from language and ontology alignment mappings, has been previously referred to as SMRT – Semantic Model-driven Representation and Transformation.

a) Formalize a metamodeling facility (e.g., MOF), including the meta-metamodel, the metamodels that conform to the meta-metamodel, the models/ontologies that are instances of the metamodels, the instances (i.e., system state or data) that are instances of the models/ontologies and the numerous relationships between these entities. The formalization must support verification and validity checking.
b) Means of formalization that is commonly applicable across the gamut of logics, formal languages and ontologies of all kinds despite vast differences in syntax and semantics.
c) Transform expressions in one language into another while conserving meaning (and the knowledge to know when this is/is not possible.
d) Define by composition a new language from existing languages or a new theory from existing theories.
e) Normalization of languages and ontologies into compositions of language and ontology components and the interrelations between them.

The three approaches we will now consider are:

- Category-theoretic formalization of a metamodeling facility (CMF - Core Metamodeling Facility)
- Institutions, Institution Morphisms, Charters and Parchments (Inst)
- Information Flow Framework (IFF)

Core Metamodeling Facility

CMF is a research effort spanning a decade, led by Ken Baclawski of Northeastern University with primary contributions by Jeff Smith, Mitch Kokar and others. See *"*Metamodeling Facilities, Work in progress for UML 2.0 Math Framework and MOF 2.0 Transformation Proposal for OMG*" Kenneth Baclawski, Mieczyslaw Kokar and Jeff Smith Sept. 2003*. The aim of CMF has been to formalize the MOF using mathematical category theory with proofs conducted in the formal language Slang in the Specware product. The strategy Baclawski et al have employed is to define a small core facility for metamodeling, formalize the core and then define MOF in terms of the core (a bootstrapping strategy).

The approach defines each layer of the metamodeling facility – meta-metamodel, metamodel, model and instance – by specifying a literal type structure(s), partially ordered sets, and order preserving functions. For example, the isAbstract quality of a metamodel element is represented as a property function with domain GeneralizableElement and range Boolean. Proofs are accomplished by category theoretic commutative diagrams and partially ordered commutative diagrams. In this fashion it is shown that CMF is both self-describing and supports the layers and interrelationships required of a metamodeling facility. The conditions that formally characterize the MOF in terms CMF are then provided as axioms; these are the MOF axioms.

This work demonstrates that a four-level metamodeling hierarchy such as MOF may be formalized using category theory with the formalization supporting verification and validity checking by means of category theoretic proofs. This work therefore supports our capability (a).

Institutions

We quote from *Institutions: Abstract model theory for Specification and programming*, Joseph Goguen and Rod Burstall, in Journal of the ACM, 39, No. 1, Jan. 1992, pages 95-146.

> We introduce the concept of *institution* to formalize the informal notion of "logical system." The major requirement is that there is a satisfaction relation between models and sentences which is consistent under change of notation. Institutions enable us to abstract from syntactic and semantic detail when working on language structure "in-the-large"; for example, we can define language features for building large structures from smaller ones [using category theoretic colimits], possibly involving parameters, without commitment to any particular logical system. This applies to both specification languages and programming languages...[Results of this work include that] any institution such that signatures (which define notation) can be glued together, also allows gluing together theories (which are just collections of sentences over a fixed signature)... gives conditions under which it is sound to use a theorem prover for one institution on [translated] theories from another... how to define institutions that allow sentences and constraints from two or more institutions. All our general results apply to such "duplex" or "multiplex" institutions.
> ...
> In particular, if we are correct that the essential purpose of a specification language is to say how to put (hopefully small and well-understood) theories together to make new (and possibly very large) specifications, then much of the syntax and semantics of specifications does not depend upon the logical system in which the theories are expressed...
>
> Informally, an institution consists of:
> - a collection of signatures (which are vocabularies for use in constructing sentences in a logical system) and signature morphisms, together with for each signature S,
>   - a collection of S-sentences,
>   - a collection of S-models, and

  erick@synsyta.com

- a S-satisfaction relation, of S-sentences by S-models,

such that when you change signatures (by a signature morphism), satisfaction of sentences by models changes consistently.

One of the results of this paper mentioned above is particularly worth expanding on for the present purposes.

> Finally, "multiplex" institutions permit whatever combination of sentences and constrains one might desire, provided they are related by morphisms to the same base institution... Signature morphisms play a basic role in structuring specifications. Let us assume for concreteness of exposition that the signatures have sorts and operators, and then consider some specific structuring mechanisms. First, we may build a more complex specification by adding new sorts and operators to an existing signature; then the inclusion of the original signature into the extend signature is an "enrichment" signature morphism. Second, we may wish to use such an enrichment not just on one specification, but on a whole class of specifications. This leads to parameterized specifications. For instantiation, the parameter sorts and operators and operators are bound to particular sorts and operators by a "binding" signature morphism. Third, a large specification may have name clashes: two subspecifications may happen to use the same sort or operator names. These can be eliminated by signature morphisms that define renamings. Enrichment, binding and renaming raise no deep logical problems, but are still important for modular structure. Using institutions, we can define such features without making a commitment to any particular logical system. Moreover, the task of giving a semantics for the language is also simplified. We feel that these considerations justify an attempt to deal with logical systems in a general way, free of the entanglements of any particular syntax and semantics.

Over its twenty-plus year existence multiple researchers have applied the theory of institutions to represent numerous logics and formal languages. Nearly every logic one can think of has been shown to be an institution. The appendix in the just cited paper provides proofs for many sorted equational logic (equations as sentences, algebras as models); first order logic (with the usual first order sentences and structures); many sorted first order logic; first order logic with equality; many sorted first order logic with equality; Horn clause logic; Horn clause with equality; many sorted Horn clause logic with equality; and others. Page 141 of the above paper lists further logics for which it seems clear the same proof methods will work, including higher order logic, standard modal logics (with Kripke structures as models) and infinitary logics; proofs for these and many others appear in the literature. The historical emphasis has been on

logics that are useful for specification. But the logics for KIF and (pure) Prolog are among those used for ontologies that have been proved institutional, leaving little doubt that RDF, OWL, etc. are also institutional.

Institutions thus demonstrate that a common formal approach can indeed be employed across a very wide range of formal systems. Furthermore, the CMF discussed in the last section has also been articulated as an institution. See *"An Institutional Framework for Metamodeling" Kenneth Baclawski, Mieczyslaw Kokar and Jeff Smith Sept. 2003*. For more on morphisms between distinct institutions – a means of transforming from one institution to another - see *Institution Morphisms*, Joseph Goguen and Grigore Rosu, in Formal Aspects of Computing 13, 2002, pages 274-307. A related approach, applying the Information Flow theory of Barwise and Seligman (one of the three cornerstones of IFF, see below) is the IF-Map approach, with demonstrated examples available at http://www.aktors.org/technologies/ifmap/. Additional examples are presented in a paper by Marco Schorlemmer, "Formal support for representing and automating semantic interoperability".

Our team, and others, have developed tools that implement parts and aspects of the theory of institutions for a variety of purposes. These include CASL and CafeOBJ (both based on Goguen's OBJ) and Maude (previously used on DARPA programs to implement complex institution morphisms).

The success of institutions across so many languages provides a measure of confidence in support of our capabilities (b), (c) and (d).

Information Flow Framework

We now look at IFF, an ongoing effort led by Robert Kent at the IEEE by which to componentize, rationalize and verify the IEEE Standard Upper Ontology (SUO) and specific domain ontologies. See http://suo.ieee.org/IFF/ . IFF employs elements of category theory, the Information Flow work of Barwise and Seligman and Formal Concept Analysis (FCA) to achieve these aims.

Of particular interest for our purposes is the IFF structure called the Lattice of Theories (LoT). The LoT formalizes a notion John Sowa calls "Knowledge Soup", an infinite, evolving structure that enables even conflicting, inconsistent knowledge to coexist and be interrelated.

To formalize knowledge soup the LoT employs FCA's notion of a *concept lattice*. The concept lattices of FCA are essentially Galois lattices formed over what FCA calls a *formal context*: a set of objects, a set of attributes and a relation that associates each object in the formal context with the attributes in the formal context it possesses. Each node of an FCA concept lattice is called a *formal*

concept and is defined by a set of objects, called an *extent*, and a set of attributes, called an *intent*, drawn from the formal context such that a Galois connection holds between intent and extent so that each attribute of a concept's intent is possessed by all of the objects in its extent; each object in the concept's extent is characterized by all of the attributes in its intent.

The LoT of the IFF is a concept lattice where each formal concept has as an extent a class of models and as an intent a closed theory (as a set of expressions in a language L) such that the extent is the class of models that satisfy the theory and conversely that the intent is the theory that is satisfied by each model in the extent.

Importantly, the partial order of the LoT is an ordering of theories: a theory T_1 that is lower down in the LoT than a theory T_2 means that T_2 is more general than T_1 (by more general we mean T_2 is constituted by a subset of the expressions that constitute T_1 – the former has fewer constraints than the later and hence is more general). Thus the LoT provides a means to interrelate theories and by following lattice edges navigate amongst theories, indeed to revise theories. In "The IFF Approach to the Lattice of Theories" http://suo.ieee.org/IFF/work-in-progress/ Robert Kent employs the LoT to formalize John Sowa's informal notion knowledge soup.

Figure 20 *Navigating the Lattice of Theories*, is from this citation showing how one can revise a theory by appropriate navigation about the LoT. For example, by moving upwards from T_1 to T_2 a contraction has been achieved (contraction referring to the reduced set of expressions (axioms) that constitute the more general theory). By moving down from T_2 to T_3 an expansion has been achieved (adding expressions to the theory so as to narrow it). Such a contraction followed by an expansion constitutes a theory revision. Additionally, by renaming the parts of axioms that constitute a theory – relation types, entity types and constants - analogous theories can be discovered.

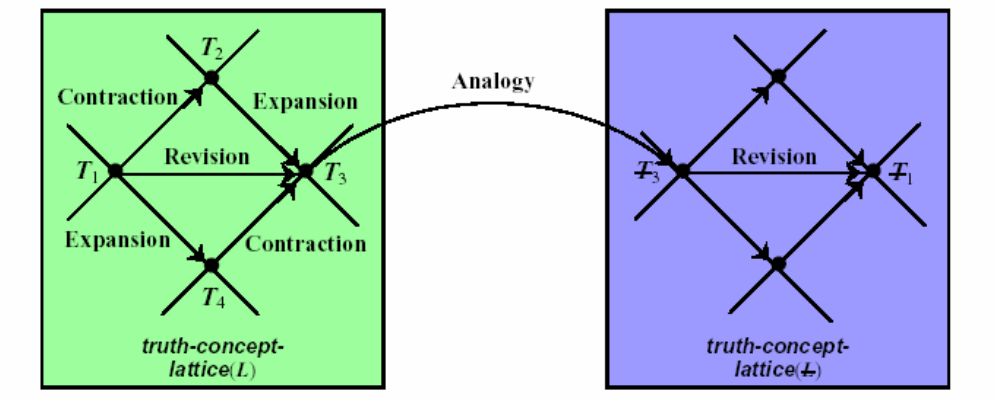


Figure 20 ***Navigating the Lattice of Theories***

This becomes clearer through an example. We recapitulate (and slightly paraphrase) John Sowa's example from his book "Knowledge Representation Logical, Philosophical, and Computational Foundations" p.387. Consider theory T_1 as Newton's theory of gravitation applied to the Earth revolving around the sun. The contraction from T_1 to T_2 involves the deletion of axioms for the gravitational force. In the expansion from T_2 to T_3 axioms for the electrical force would be added. The net revision of T_1 to T_3 is to replace the gravitational force in what is the Copernican model of the solar system with the electrical force. Finally, we jump via analogy to a remote theory in the LoT (the previous movements have all been local). For this analogy we systematically rename the type, relations and individuals that appear in the axioms: the Earth is renamed the electron; the sun is renamed the nucleus; and the solar system is renamed the atom. A final revision of this analogy can discard details about the Earth and Sun that have become irrelevant and add new axioms for quantum mechanics.

Another facet of the LoT, important to our present endeavor though not elaborated on by Kent in the cited paper on IFF, is construction of theories. The standard lattice operations of join (supremum – least upper bound) and meet (infimum – greatest lower bound) provide the means to form new theories from existing theories. (Note: By treating the LoT as a full-fledged mathematical category additional options for theory creation become available.) Particularly intriguing are the theories in the LoT that are minimal with respect to the axioms that constitute them. Such theories may serve as atomic components for composing more elaborate theories in the same vein as when Goguen speaks of using institutions to "put (hopefully small and well-understood) theories together to make new (and possibly very large) specifications".

In the opening of this subsection on IFF we remarked that the purpose of IFF, including its LoT, is to componentize, rationalize, verify and even merge ontologies, both the general-purpose SUO as well as far more specific domain ontologies. The central tactic here is that the IFF treats *ontologies as theories*. The consequence is that the LoT may be employed to componentize, merge, revise and generally navigate ontologies; indeed, to compose ontologies from ontology components! And having composed two or more distinct ontologies (from a library of ontology components) to know where the composed ontologies are situated in the LoT and use the LoT to navigate – in other words, map or transform – from one ontology to another without further adieu.

The IFF is focused on solving ontology problems through application of its mathematical framework at what we call (in MOF speak) the M1 level, the level of models/ontologies. However, we believe that there is great benefit to be had by additionally applying IFF at a higher meta level, specifically M2, the level at which metamodels are used to represent, manipulate, manage and transform

languages. We therefore take this further step and treat *metamodels of logics and languages as theories.* We conceive of an IFF-like LoT where the theories represent the metamodels of logics and languages, including metamodel components of logic and language. This LoT at the metamodel level provides the means to compose metamodels of logics and languages from metamodel components of logic and language, and to navigate amongst such composed metamodels of logics and languages. Add this to the analogous capability for ontologies just discussed and we have the means to capability (e).

Where we go from here

In the remaining section we take a very early look at a notional strategy for component-wise normalization and recomposition of languages, logics and ontologies along the lines enabled by CMF, Institutions and IFF. For convenience of exposition we introduce the term synthetic language systems (SLS) to designate non-natural languages, logics and formal systems and their artifacts across all metalevels (including meta-metamodel, metamodel, model/ontology and instance).

Given the mathematical and computational tools and technologies described throughout this paper we now seek to identify the mostly orthogonal concerns by which an arbitrary synthetic language system may be factored (i.e., normalized to components). We have provisionally defined a (mathematical) concern space of seven dimensions to accomplish this. The dimensions are: logic constructors, ontology constructors, abstract syntax, representational idiom, axiomatic semantics, model theoretic semantics and proof theory. Points and regions of this space may be mapped to and from a multitude of concrete and surface syntaxes (which may also be considered as an aspect of the syntax dimension of concern) in much the same way as MOF manages artifacts at all metalevels in terms of an abstract syntax but may externalize to and internalize from specific surface syntaxes.

There must additionally be a scheme for defining elementary components along each identified dimension of concern and for interrelating the larger-grained, composed components, e.g., the elementary logic constructors and the structure that interrelates compositions of these up to the complexity of a complete language/logic. We have outlined a scheme for defining elementary and composed components for each of the seven dimensions of the concern space. The schemes draw on the work referenced above to formalize and componentize synthetic language systems.

A central problem to the effort is composing transformations between synthetic language systems from more foundational mappings, and deriving these from

the relations that hold between their constituent components. From a category theoretic viewpoint this involves producing morphisms from morphisms. The work of team member Joseph Goguen with Ron Burstall on Institution charters and parchments looks to be a promising foundation for such, and these structures have already been successfully used by Mossakowski and others on problems in the semantics of Institutions with tools like CASL.

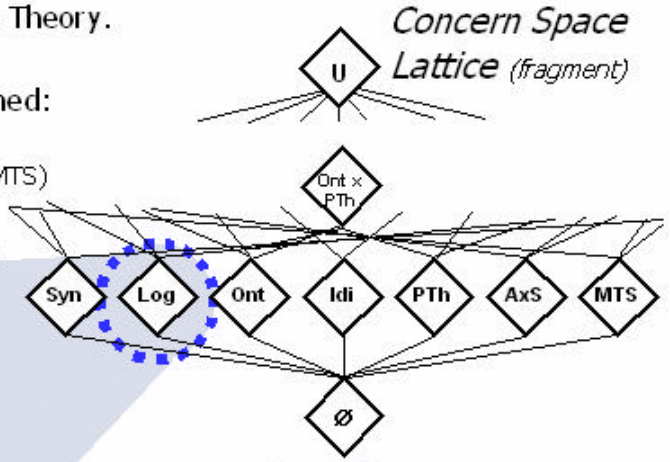
MAGIC - Managed Logic - Technical Approach

Formal Underpinning

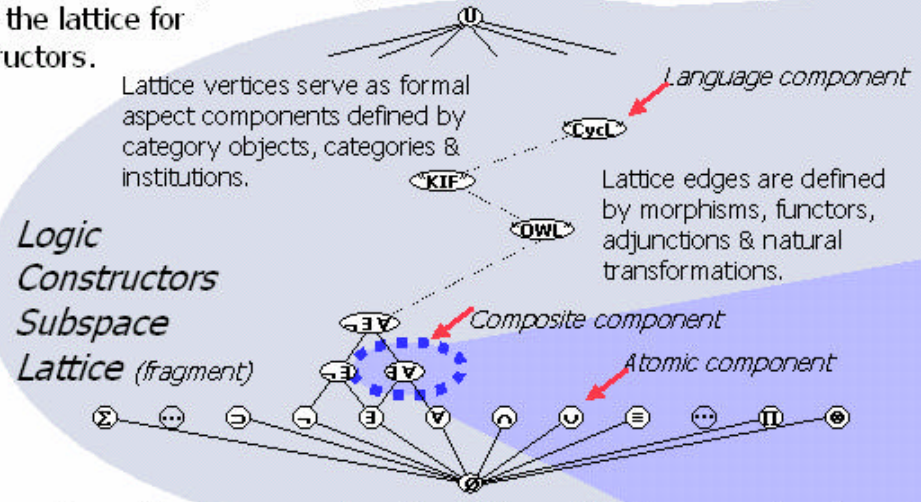
The formal underpinning embraces a flavor of aspect orientation known as Multidimensional Separation of Concerns (MDSOC) and formalizes it using Category Theory and Lattice Theory.

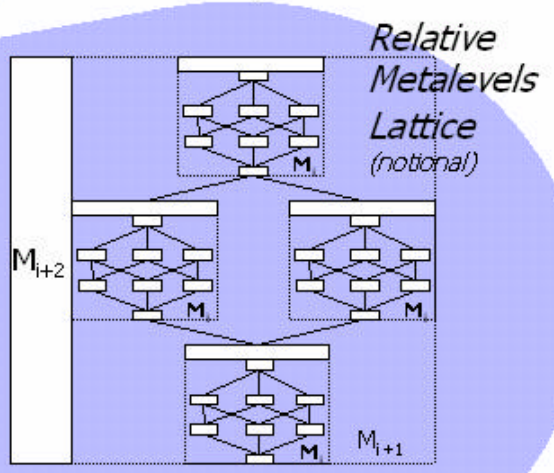
A Concern Space Lattice (shown to the right) represents the subspace structure of a multidimensional space spanning seven dimensions of concern by which SLS are defined:

- Abstract & surface syntaxes (Syn)
- Logic constructors (Log)
- Ontology constructors (Ont)
- Representational idioms (Idi)
- Proof theories (PTh)
- Axiomatic semantics (AxS)
- Model theoretic semantics (MTS)



Each subspace is in turn represented by a Concern Subspace Lattice; below we see a fragment of the lattice for Logic Constructors.





Category theoretic methods are employed to define a SLS by composing formal aspect components from within and across the concern subspaces. For example, the complete definition of OWL-DL would span vertices from Syn, Log (seen above), PTh & MTS.

These formal compositional definitions preserve traces to their constituents and conjoining tissue. Such traces form the basis for deriving syntactic and semantic transformations between SLS.

The formal underpinning is itself defined in the facility, enabling it to be verified, extended, transformed and migrated.

Associated with each concern subspace is a nested lattice structure componentizing language metalevels, e.g., language, model / ontology & instance.

The figure on the previous page (MAGIC – Managed Logic – Technical Approach) summarizes our formal framework in support of the recursive process we've described in this paper to provide largely automated interoperability and information flow between synthetic language systems. A thorough walk-through of this figure is beyond the scope of the present paper. We do however, direct attention to several highlights. Notice the Concern Space, Subspace and Relative Metalevels lattices, strongly influenced by IFF's notion of the Lattice of Theories (though flipped top-to-bottom). Observe that the framework operates on all metalevels, that these have been made relative, and that the framework is self-describing. Attend to the use of category theoretic methods by which to compose big things from small things, derive mappings between compositionally defined entities and articulate proofs.

For the B2B example the relational model of data and XML and XML Schema would need to be compositionally defined in the framework as languages and OBI and UBL as ontologies would similarly need to be defined. We believe the methods discussed in this section may then be used to automatically derive the language mapping between relational and XML as well as to derive the ontology alignment mapping between OBI and UBL. From these mappings, which would be persisted for future reuse, we would reason the way to a transformation model as discussed in the previous section and then interpret or compile the transformation to produce the target XML representation of the line item.

Despite such advanced mathematical machinery there will doubtless be cases that resist full automation and require human intervention. We suspect such cases will be far more likely when deriving ontology alignment mappings than when deriving language/logic mappings because we expect the definitions of languages and logics to have less inherent ambiguity, variability and complexity than the definitions of ontologies. In such cases humans must be empowered with visual/graphical syntaxes for ontology alignment and high-level tools that support them. Our brief review of graphical techniques for ontology definition, management and alignment portends a direction, but considerable work is needed to simplify the process so that domain specialists find it straightforward to achieve good results with high assurance.

At the base of our recurse we strike what for us is bedrock: the definitions of logics, languages and ontologies themselves in terms of atomic components. Once again, we see a need for visual/graphical methods and tools for authoring and management of the language definition process. Such methods and tools must handle the arcane mathematics under-the-covers for the domain user while enabling the expert full, unfettered access and control.

Appendix 1: MOF based metamodeling

A metamodel is a model that describes and structures another model.
In the case of our illustrative example, the relational model of data is the metamodel for the purchase order line item schema, i.e., the line item schema conforms to the concepts, structures and constraints of the relational model of data – line items are represented as rows in a table; each line item must have a unique key; the table of line items must not contain duplicate rows; the rows of line items in the table are unordered; the columns of the line item table are unordered; etc.

A metamodel may represent a language, i.e., its concepts, structures and constraints.

In terms of our example, the "relational model of data" may equally be called the "relational metamodel".

Now come some thought-provoking questions: If a metamodel is a model that describes and structures another model, then a metamodel is a model in its own right. A model is expressed in a language, so what language is a metamodel expressed in? Doesn't this create an infinite regress, since the language used to express the metamodel must itself be defined a priori?

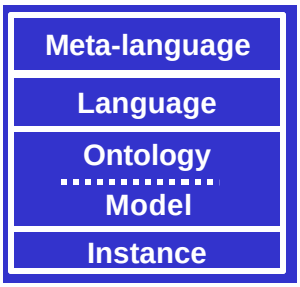
Let's approach this question in a somewhat different order than it is phrased. A metamodel is indeed a model in its own right, and must therefore be expressed in a language. In the case of the relational metamodel, the language it is defined in is the mathematical language of set theory. This itself begs two more questions: "Is there a formal role for the language with which a metamodel is expressed?" and "In the case of the relational metamodel, is mathematical set theory some kind of meta-metamodel, and if so, in what language is set theory defined?

There is indeed a formal role for the language with which a metamodel is expressed, and if it is expressed in the form of a model it is called a "meta-metamodel". Mathematical set theory is the meta-metamodel typically used to define the relational metamodel.

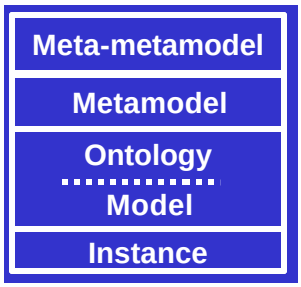
However, mathematical set theory is the end-of-the-line, containing a small set of axioms that are assumed to be true, with the rest of set theory consisting of constructions, theorems and corollaries that logically follow from the given axioms.

  erick@synsyta.com

(Aside: Some theoreticians in the mathematical discipline known as Category Theory utilize a special formal *category* to define set theory, making Category Theory, or at least a part of it, the new end-of-the-line. For the rest of the current discussion we will treat set theory as the stopping point of the representational regress.)



We can graphically summarize these thoughts, presented to the right. This structure will be quite familiar to those steeped in the world of modeling and metamodeling, particularly as these are embodied in the standards of the Object Management Group.

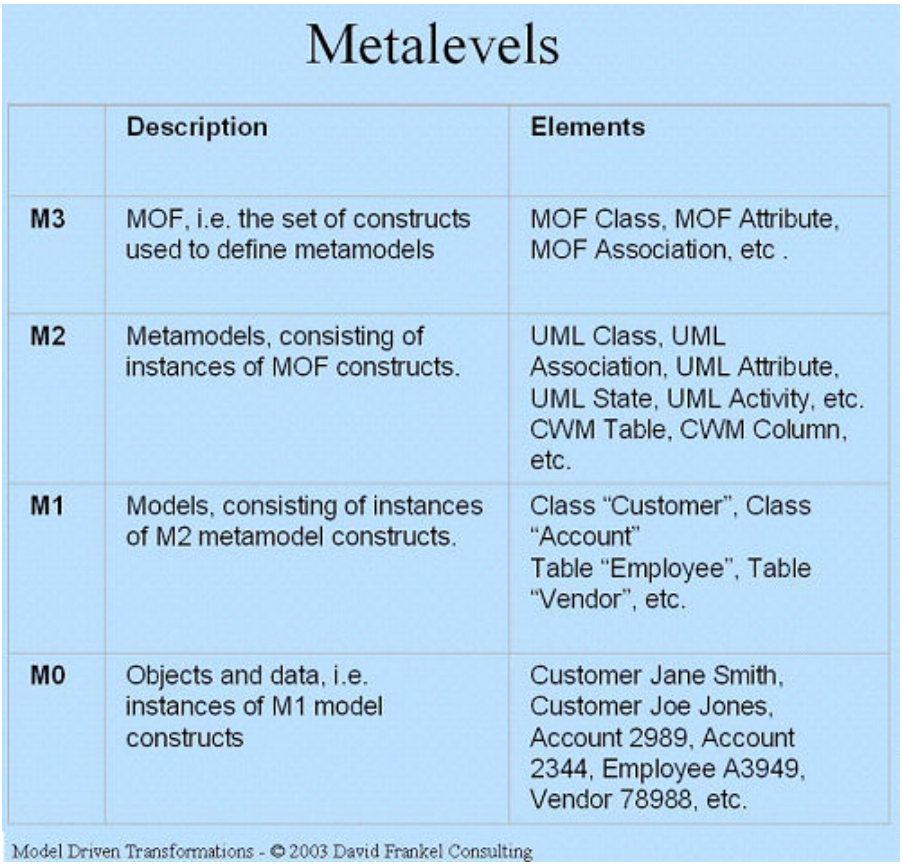


For our immediate purposes we designate the meta-metamodel as the end-of-the-line, preventing an infinite regress by expressing the meta-metamodel in itself, i.e., the meta-metamodel is structured by and upholds the same principles it imposes on those metamodels defined with it. To put it another way, the meta-metamodel is defined and expressed by its own concepts, structures and constraints.

Let's begin putting some tangible flesh on this skeletal discussion that has grown increasingly theory-laden. The UML standard, Unified Modeling Language, is defined as a metamodel, expressed in terms of a meta-metamodel called the Meta Object Facility, or MOF for short.[7] Thus, UML the modeling language is defined by MOF concepts, structures and constraints. However, UML is not the only MOF conformant metamodel. There are many; some are adopted standards.

[7] UML and MOF are standards of the Object Management Group.

When discussing the various layers of representation in an OMG context we often designate these as metalevels M0-M3, as is shown in Figure 21.



Metalevels

	Description	Elements
M3	MOF, i.e. the set of constructs used to define metamodels	MOF Class, MOF Attribute, MOF Association, etc .
M2	Metamodels, consisting of instances of MOF constructs.	UML Class, UML Association, UML Attribute, UML State, UML Activity, etc. CWM Table, CWM Column, etc.
M1	Models, consisting of instances of M2 metamodel constructs.	Class "Customer", Class "Account" Table "Employee", Table "Vendor", etc.
M0	Objects and data, i.e. instances of M1 model constructs	Customer Jane Smith, Customer Joe Jones, Account 2989, Account 2344, Employee A3949, Vendor 78988, etc.

Model Driven Transformations - © 2003 David Frankel Consulting

Figure 21 *MOF Metalevels*

(Aside: Not to cause confusion, MOF itself reuses a carefully chosen subset of UML. One may then conceive of the entirety of UML as defined and structured by a small core of UML, which is used to define and structure itself.)

In Figure 22 *UML Metamodel*, we see a fragment of the MOF conformant UML metamodel for "class".

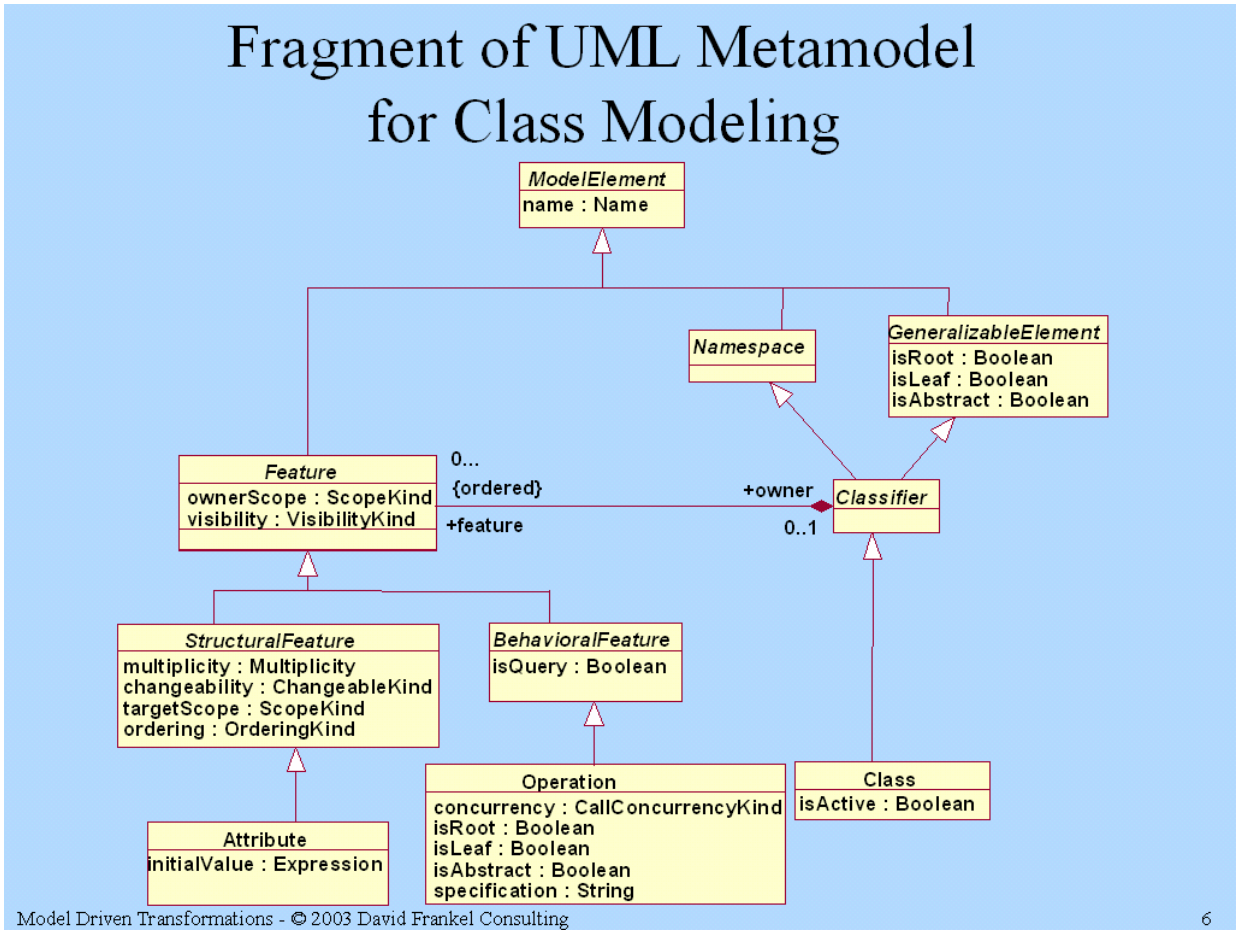


Figure 22 *UML Metamodel*

We can immediately see that the UML construct Class, as well as the constructs it contains, Attribute and Operation, are specializations of Classifier and Feature, which in turn are specializations of ModelElement. These last constructs – Classifier, Feature and ModelElement – are MOF constructs; in other words they are meta-metamodel elements used to define the UML metamodel.

MOF metamodeling has intrinsic benefits that aid in the definition of a language, aside from the support MOF supplies for automating transformations. And UML is not the only language to be represented by a MOF metamodel. To understand these benefits, consider the ISO activity in progress to define a predicate language named Simplified Common Logic (SCL). SCL will be the successor to the ISO Knowledge Interchange Format (KIF), and is based on the Common Logic work of John Sowa and others. The authors expect SCL to be an important part of the Semantic Web.

The primary author of SCL is Pat Hayes, an expert in formal logic and one of the authors of the Semantic Web specifications. Pat has been writing an SCL specification document that defines SCL's abstract syntax, semantics, and a textual concrete syntax for the language. He has simultaneously participated in an activity with some experts in Model Driven Architecture to define a MOF metamodel for SCL. The metamodel captures SCL's abstract syntax as a formal MOF model. This is a case where, instead of defining the MOF metamodel for a language retrospectively—that is, after the language has already been defined and come into use—the MOF metamodel is being defined concurrently with the process of defining the language.

The process of creating a MOF metamodel of the SCL abstract syntax has helped with the definition of the language, surfacing errors and raising issues that might have escaped notice otherwise. As is typically the case for MOF metamodels, the SCL metamodel uses UML notation, and formally states invariant rules pertaining to the abstract syntax. The visual model helps to make the abstract syntax more intellectually manageable; this, combined with the process of writing formal invariant rules, tends to "shake out" bugs in the language.

Furthermore, the metamodeling team, including Pat Hayes, Elisa Kendall, David Frankel and Deb McGuinness, used a model compiler that is part of the Eclipse Modeling Framework (EMF). The compiler implements XMI's MOF-to-XML mapping (see Appendix 2, below), and thereby generated an XML Schema for SCL, which essentially constitutes an XML-based concrete syntax or serialization format for SCL. The ISO may be able to use this Schema rather than having to laboriously hand craft an XML Schema. The model compiler also generated code for serializing SCL expressions in and out of Eclipse, using the XMI-based Schema.

Appendix 2: Model Management with MOF and XMI

The approach of Model Driven Architecture requires that MOF conformant metamodels be authored for each transformation source and target. MOF conformant metamodels are typically authored, viewed and edited in a graphical UML modeling environment, such as IBM's Rational Rose and XDE. For the time being a full featured and complete environment requires a client-side application.

What is less commonly known is that MOF conformant metamodels may also be created, viewed, queried, updated and deleted within a *MOF repository*. Think of a MOF repository as a multi-level database for managing the MOF meta-metamodel, MOF conformant metamodels (such as the ones we've been discussing for UML and CWM), models that conform to these metamodels, and instance data that conform to the models – managing artifacts at and across all four meta levels. This can be seen in Figure 23.

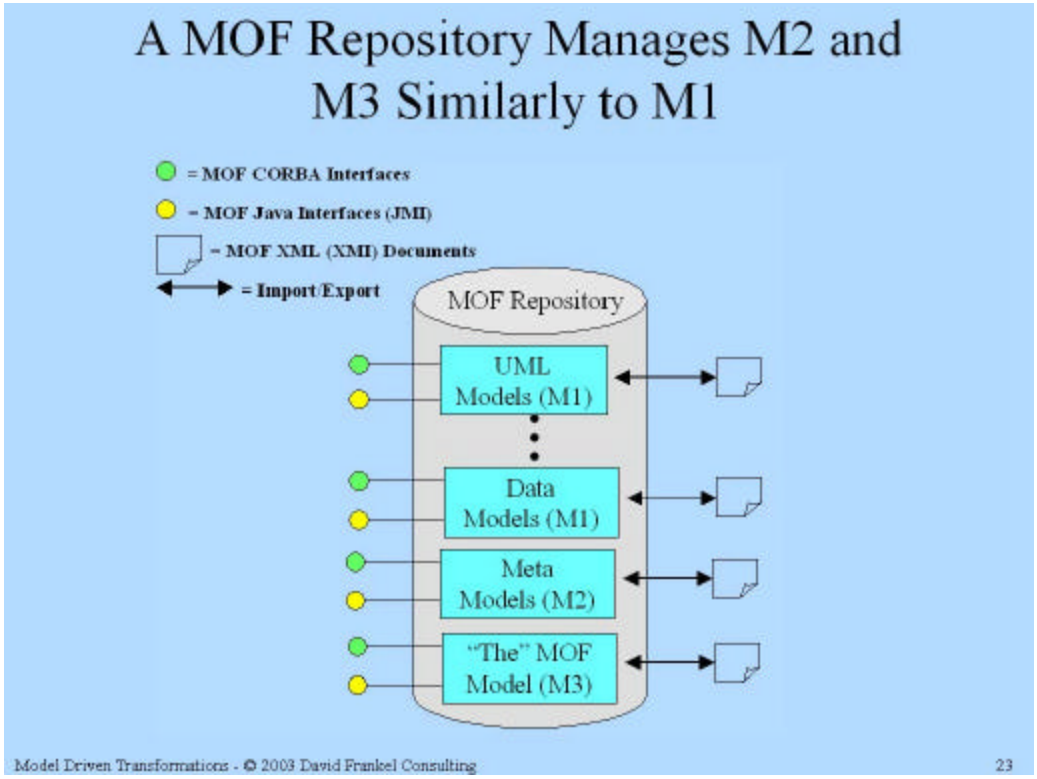


Figure 23 *MOF Repository*

Contrast this with a typical RDBMS that offers a single fixed data abstraction – everything is a relation – the user of which may only define relational schemas and populate them with relationally structured data, even if the user is the database administrator with all the powers associated with this role.

A MOF repository can certainly be used for relational information: by importing a relational metamodel into a MOF repository one can now manage relational schemas, and can even manage relational data, i.e., rows, that conform to the schemas.[8]

The crucial point, however, is that the very same MOF repository that is managing relational schemas and tables can at the same time be managing hierarchical schemas, multidimensional star schemas, XML schemas, ontological concepts and conceptual relationships... - anything for which a MOF conformant metamodel may be defined. Figure 24 *Integrated MOF Repository*, provides just a rough idea of this versatility, flexibility and representational power.

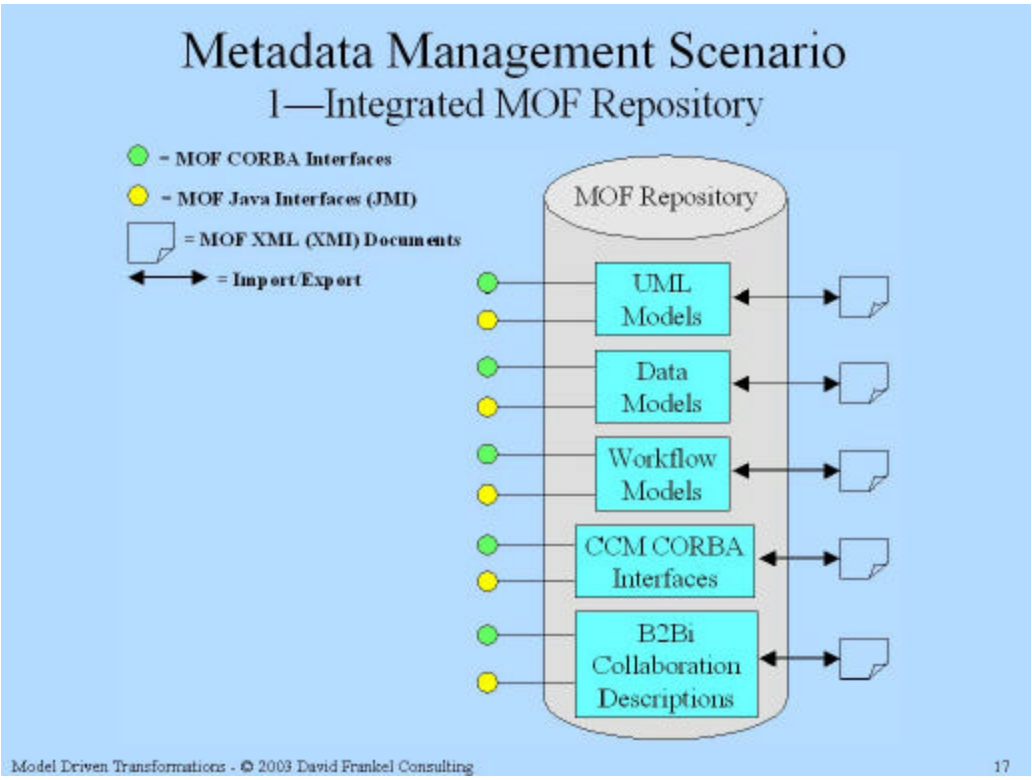


Figure 24 *Integrated MOF Repository*

Notice in this illustration the numerous modalities with which to interface and access a MOF repository, and these are just the modalities that have been standardized thus far. Two standardized programmatic interfaces are available: a Java API, known as JMI for Java Metadata Interface[9] and a CORBA interface[10]

[8] We are not recommending that large *operational stores of relational instance data* be managed in a MOF repository in this fashion – they would be better managed in an RDBMS in order to benefit from the specialized physical schemas and optimizations incorporated in products that have been developed specifically to support relational data access and manipulation.

[9] JMI is a specification of the Java Community Process, specifically JSR 40.

[10] The MOF Corba interface is a standard of the Object Management Group.

providing MOF support to the many languages supported by CORBA. Interfacing with a MOF repository may also be accomplished in an XML document-centric fashion, supported by XMI, XML Metadata Interchange[11], of which we will have more to say shortly.

In passing we note that a modeling tool may offer support for these same interfaces and modalities without providing the persistence and server-centric features of a full repository. IBM's Eclipse Modeling Framework and Sun Microsystem's NetBeans are such environments, enabling metamodels and their conforming models and instances to be accessed and manipulated programmatically and input/output via XML.

Since the transformation models we've been discussing are themselves models that conform to a MOF metamodel - the CWM transformation metamodel - these too may be defined, revised and generally managed within a MOF repository. This is shown in Figure 25 *Managing CWM Transformation Rules*.

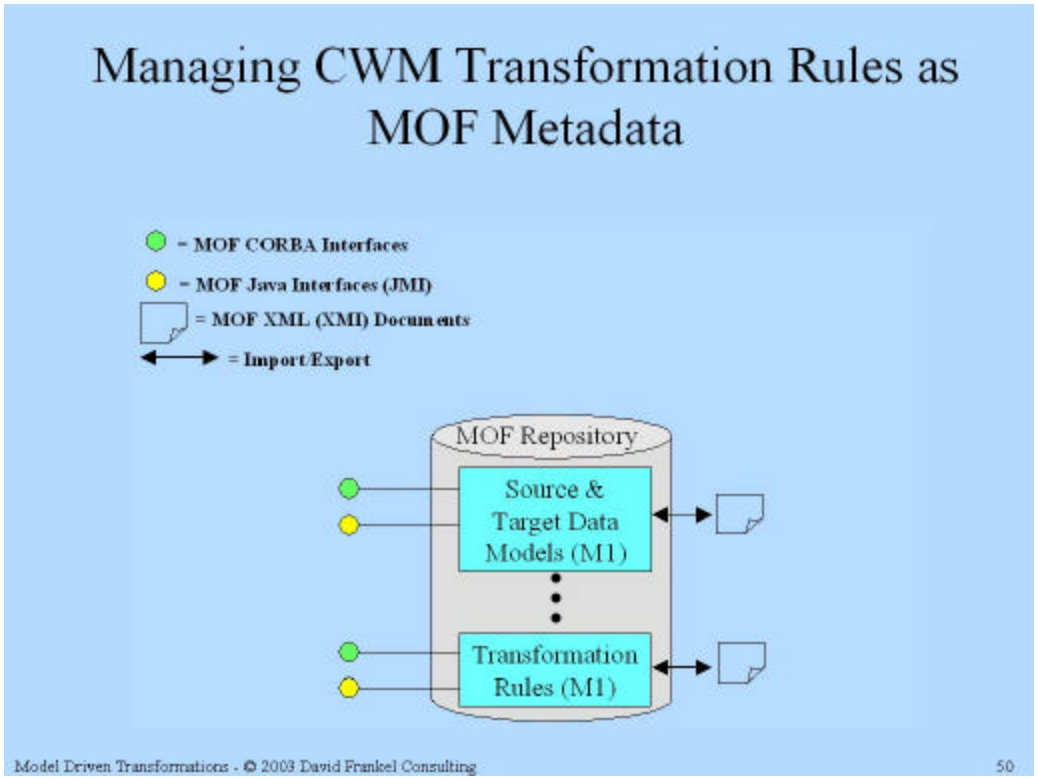


Figure 25 *Managing CWM Transformation Rules*

Additionally, MOF repository implementations (e.g., commercial products), such as Adaptive's ITPM, also support a web services interface utilizing WSDL descriptions and SOAP bindings, as well as an HTTP/HTML interface. This means that MOF conformant metamodels, models and instances can be created and

[11] XMI is a standard of the Object Management Group.

managed on the server-side, within a MOF repository, through a web service or even a thin client. Adaptive's implementation even offers SVG (Scalable Vector Graphics) views of UML and MOF models, as well as browser-based model creation and editing through forms.

There's little doubt that client-side UML environments such as Rational Rose are most appropriate to serve as the primary means of authoring metamodels and transformation models. However, these tasks are unlikely to be fulfilled by individual developers working in isolation. Rather, this will become a collaborative process, involving the contributions of more than a single individual. A MOF repository provides the means for teams to collaboratively author, review and revise metamodels and transformation models, with some team members working through a tool like Rose, publishing to and retrieving from the repository; other members will work more in a review capacity, accessing the repository, viewing its contents through a browser and suggesting revisions; while other members will tweak and rev metamodels and transformation models through a forms interface and/or a web service. A MOF repository can support the collaborative team operating in all of these diverse modes while maintaining consistency.

Such collaborative teams may be widely distributed, even working in a peer-to-peer fashion; it's important to note for operations of this type that a MOF repository needn't be configured as a centralized installation. Figure 26 illustrates a (simple) federation of MOF repositories. Far more elaborate schemes may be configured.

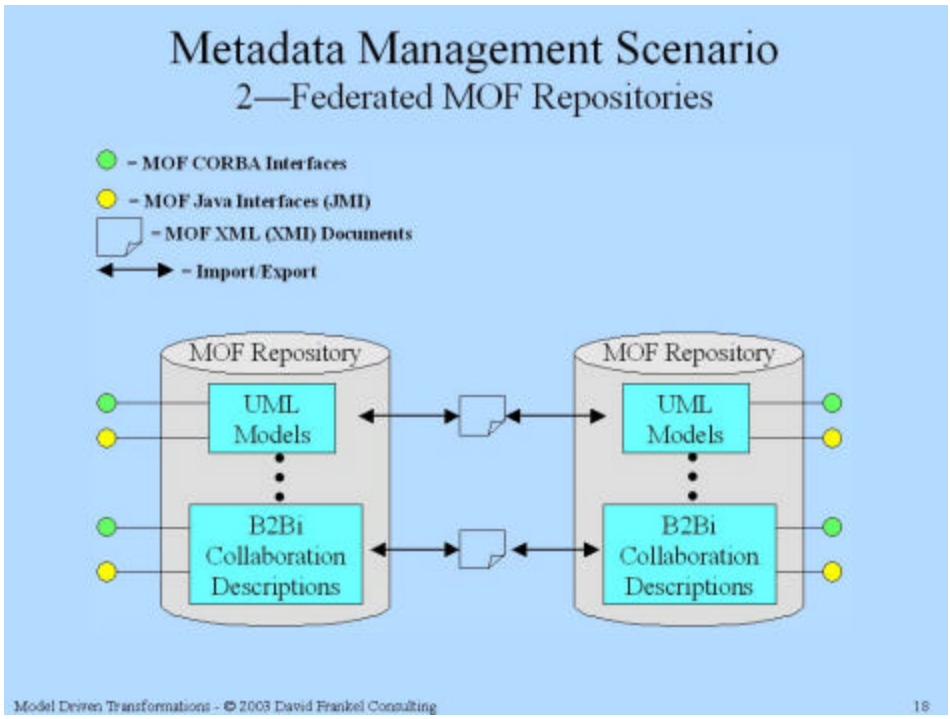


Figure 26 *Federated MOF Repositories*

Within a federate - a single MOF repository that's part of a federation - clients may access and interface the repository through a programmatic interface (which could well include a web services interface). However, the connecting fabric between federates - that which constitutes the federation - is far more likely to be embodied by XML document exchange with its tolerance for unreliable networks and its friendliness towards loose coupling. This brings our discussion to the topic of XMI, which is covered in the next section.

Serialization via XMI

Basing metamodels on the MOF meta-metamodel pays additional dividends when it comes time to externalize such metamodels, and the models and instances that conform to them, beyond the environment in which they reside or were created, e.g., a MOF repository or UML modeling environment. We've already seen that the MOF's APIs allow metamodels, models and even instances to be programmatically created, interrogated and managed. Beyond programmatic interfaces MOF also enables a modality by which metamodels and their conforming models and instances may be exported and imported as XML documents that conform to an XML DTD or XML Schema.

This capability, called XMI for XML Metadata Interchange, is provided by a standardized transformation mapping MOF to XML, as seen in Figure 27.

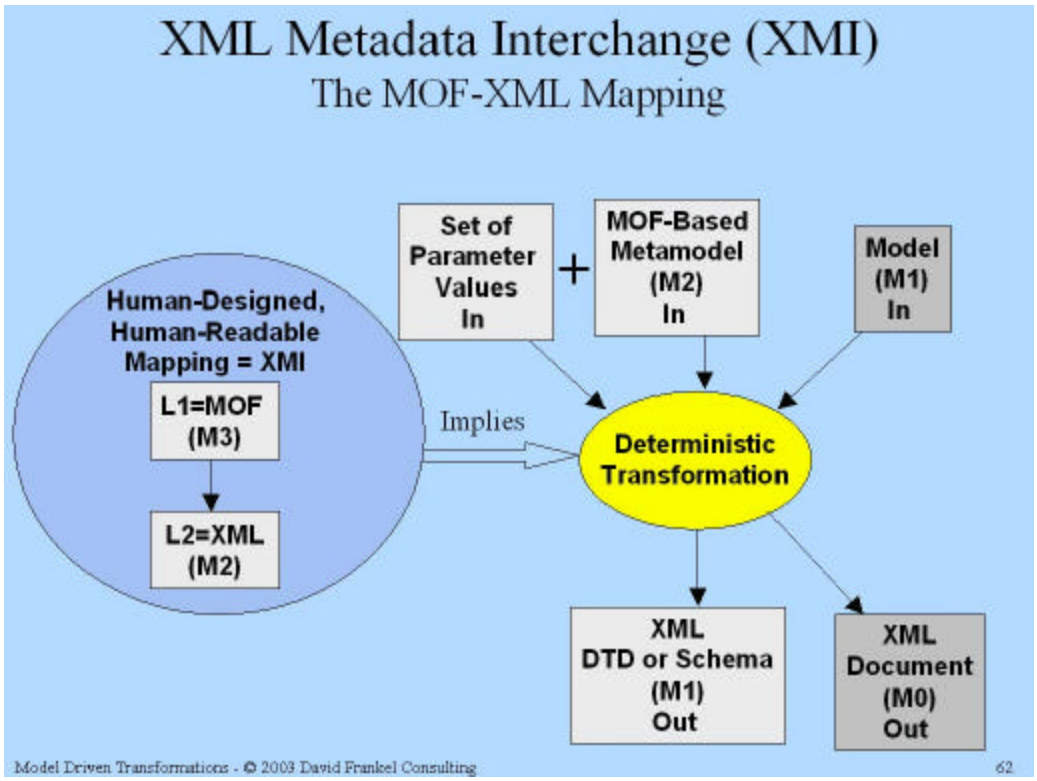


Figure 27 ***MOF-XML Mapping***

We read this diagram as follows. The standardized XMI mapping takes as input a specific MOF conformant metamodel, a specific model that conforms to the metamodel, and a set of parameter values. The product of this transformation is an XML document that represents the specific input model, now in the form of XML, plus an XML DTD or XML Schema that represents the specific metamodel, and to which the produced XML document conforms. This becomes clearer when we look at an example.

In Figure 28 we see the case where the inputs are the MOF conformant metamodel for UML along with a specific UML model, e.g., a UML class model for the Employee class. The output is an XML document that represents the Employee class model, and an XML DTD or XML Schema that represents UML, to which the XML Employee document conforms.

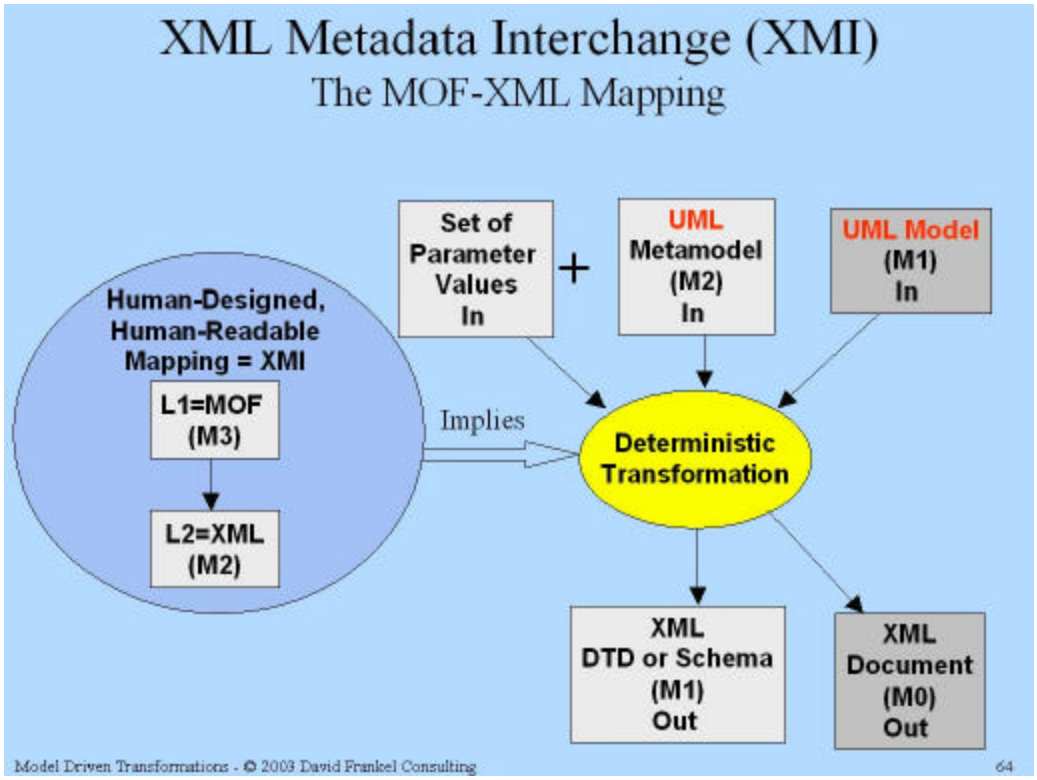


Figure 28 ***MOF-XML Mapping Applied to UML***

It is a common misconception that the XML DTD or XML Schema of this last example (that represents the MOF conformant UML metamodel) is the total extent of XMI. Though highly useful, this DTD and Schema is but one artifact that the XMI mapping can produce. XMI will produce a DTD or Schema for any MOF conformant metamodel. Examples include DTDs and Schemas for the CWM metamodels we looked at earlier, such as the relational metamodel, and any custom defined metamodels, such as ones that may be created for the information products of analyst tools and services. Even transformation models can be serialized as XML using XMI. The fact is XMI provides a common serialization/deserialization across all metalevels. This is seen by example in Figure 29.

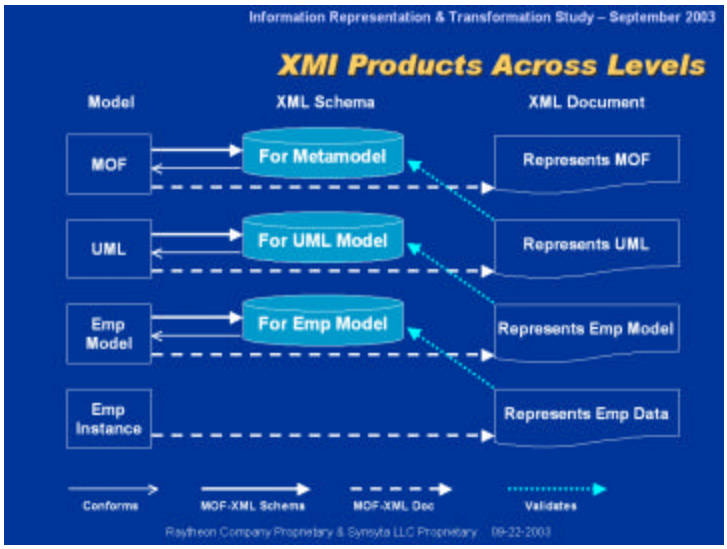


Figure 29 *XMI Products Across Levels*

Export via XMI is one side of the equation, but it also supports import. These transformations can essentially be run in reverse order, taking an XMI XML document as input and recreating the original model. Some implementations of XMI that support this process do not even require the XML DTD or Schema when performing the import – they literally read the metamodel, rather than the DTD or Schema, and use it to parse the XMI document.

There's another interesting twist on the process, this one also regarding XML as input. The XMI standard will also create a model when the input document is XML but not XMI compliant. This is shown in Figure 30. In other words, if one has an existing XML DTD or Schema, created by any means, one may use the XMI machinery to reverse engineer the model that is implicit within. One must provide or specify the metamodel that this model will conform to; this is typically the UML metamodel, but it needn't be.

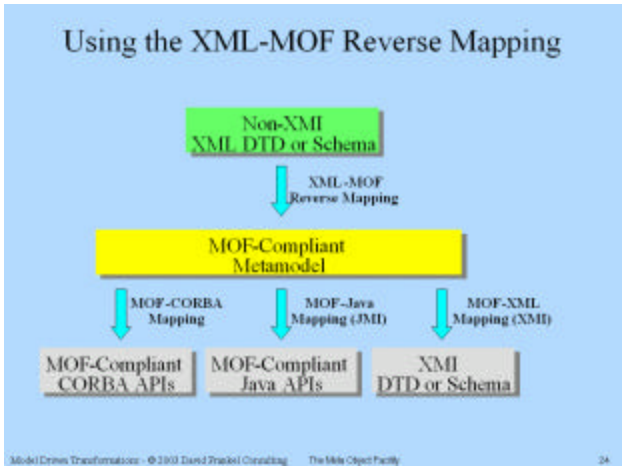


Figure 30 *XML-MOF Reverse Mapping*

  erick@synsyta.com

While it should be noted that non-XMI XML lacks some of the information captured by XMI XML, and thus reverse engineering may not produce as rich a model as if one had created it in a UML modeling environment in the first place, this reverse engineering provides an excellent starting point for model development. Consider the notional process below.

1. Start with an existing XML DTD or Schema (to which existing instance data, represented as XML documents, conforms)
2. Automatically reverse engineer its UML model (or model that conforms to a different metamodel)
3. Revise and elaborate the model
4. Apply forward generation to produce a new DTD or Schema that conforms to the elaborated model
5. Define a transform between the old, reverse engineered model and the new elaborated one
6. Execute the transform to move XML document data to the new elaborated model as represented by the new XML DTD or XML schema

XMI is a flexible and agile means for moving metamodels, models and instances between repositories and tools of all sorts.

  erick@synsyta.com