Development of Open Source E-Gov Reference Architecture (OsEra) Prototype

Metadata Infrastructure Architecture

General Services Administration (GSA)

Office of the Chief Information Officer (CIO)

Draft: September 3, 2005

Table Of Contents

Overview

The OsEra Metadata infrastructure will provide a "smart" repository for architectures at multiple levels of abstraction, from multiple sources and with multiple views. This infrastructure will integrate the OMG-Meta Object Facility (MOF) and Resource Description Framework (RDF) and RDF-Schema as defined by W3C as part of the "Semantic Web" initiative and will integrate the specification and provisioning concepts of the OMG Model Driven Architecture (MDA).

Problem Statement

Information about government processes, information and I.T. systems currently exists in a diverse set of forms, formats, repositories and documents with little to no management and coordination. The result is that there is rampant redundancy in the developing and re-developing the same information, different models, architectures and studies about the same things. Understanding and integrating this wide Varity of information is almost impossible and thus generally such understanding and integration is not achieved.

Not only is this information expensive and time-consuming to develop, re-analyzing the same area results in inconsistent designs, lack of interoperability, redundant systems and missed opportunities for improvement.

OsEra will replace this fragmentation and loss of information with an architected approach to developing, integrating and managing information across the government.

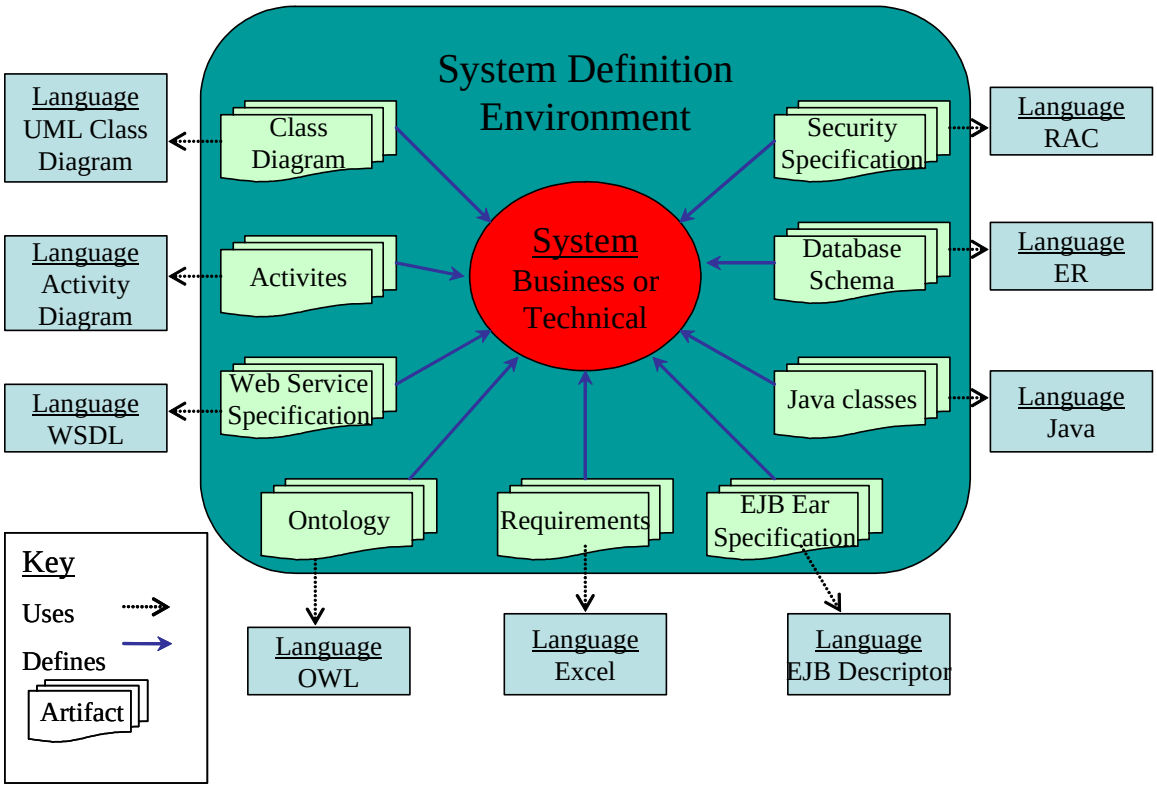
OsEra will provide this architected approach by bringing together a suite of open source technologies to provide for architecture creation, integration and management. This suite will utilize Semantic Web and Model Driven Architecture (MDA) technologies to derive value out of architectures through the automated production of specification, implementation and testing artifacts to automate the enterprise based on business requirements.

Central to OsEra are the concepts of a system and an architecture. A system is any complex set of resources and behaviors integrated for a purpose. An organization is a kind of system, a business process is a system, as is a computer application. Architecture is a specification for how a system may or will fulfill its purpose. Architectures exist at many levels, from the architecture for a new venture to the architecture of a physical part in a machine. Understanding the connections between various systems and architectures is the means for more effectively realizing the goals. Architectures are expressed in models and ontologies (the term "model" and "ontology" are used interchangeable as expressions of architectures but ontologies are also used for other purposes thus we generally use "model" to include specification of an architecture).

Systems defined by multiple languages

Modern enterprises, governments and computer systems are complex; they are complex in terms of what they do, how they do these things and on how they use various technologies. Due to this complexity systems are defined using a set of languages, each language is used to prepare various models that, together, define the system.

The following picture illustrates how a set of specifications in various languages may serve to define a system.



Our challenge, as system developers, is that these languages overlap and the specifications are **always** inconsistent. The same element in the system may be represented in multiple specifications in multiple languages. Understanding the whole system becomes difficult and error prone. The connection between these languages is often lost as the system definition evolves over time. In fact, it is rare that the entire system is understood by anybody or documented in one place.

With each language being independently conceived and designed from its own viewpoint, there is no way to understand or refer to the touch points between the languages – whether they are saying the same thing or are stating unique facts about the same thing. This lack of integration is a source of complexity and error in our systems and the reason behind many system design failures.

Compound the above with multiple systems from multiple vendors and contractors with multiple architectural approaches. *The fragmentation of information is profound.*

Languages and sources of information

OsEra will be used to manage, integrate and create architectures from multiple sources and using multiple tools based on different languages. There is no assumption that the OsEra model or infrastructure will be required or suitable for all purposes. To this end,

OsEra must be able to deal with architectures expressed using differently languages and idioms. The languages of most concern for architectures are;

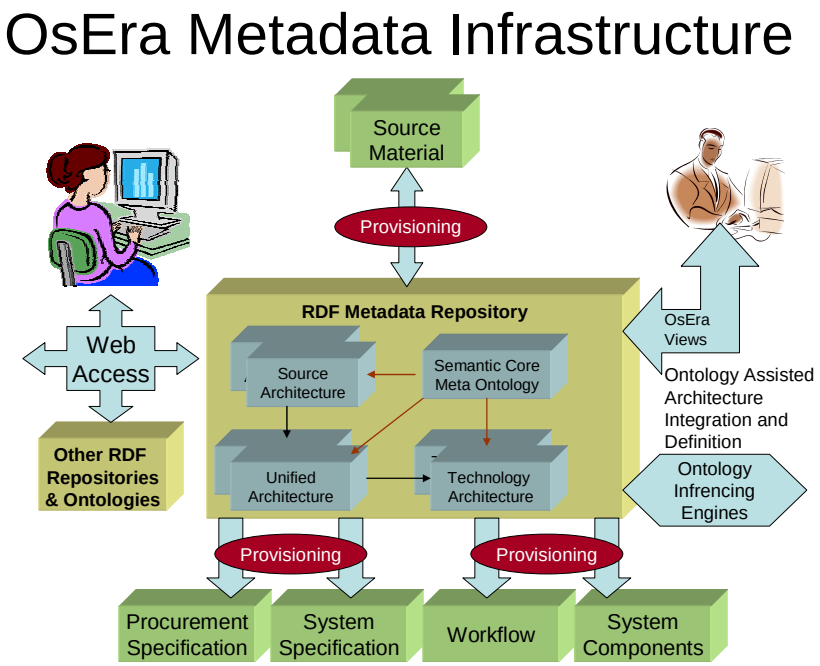
- OMG-UML-2 (Particularly class, activity and sequence diagrams)
- OWL (Actually, ontologies expressed in OWL)
- DoDaf
- Federal Enterprise Architecture
- OMG-Edoc
- BPMN
- Entity Relational
- The XML technologies – XML, XML Schema, WSDL and BEPL
- Zackman Framework
- Various proprietary tool formats. E.G. System Architect, Component-X, Rational Rose, DOORS, Metis, Etc.

To this end OsEra will be required to understand the semantics embodied in architectures expressed using these formalisms and make sense of them *together*, as a consistent view of the enterprise.

OsEra Metadata Architecture

The OsEra Metadata management facilities provide for the definition, understanding, manipulation and management of architecture models. It fits into and supports the overall OsEra Architecture and capabilities.

There is currently duplication of capability between the OMG-MOF infrastructure and the RDF infrastructure. Due to the requirement to interface with systems in both environments we intend on bridging these infrastructures such that our models may be used from either view. Ultimately one or the other will probably be selected as "core" and the other a mapping. But at this time they exist as peer infrastructures. As we refer to our metadata infrastructure we are including both the MOF and RDF interfaces to the models.



The Metadata infrastructure is the hub into which the various capabilities of OsEra integrate. This includes the 2-way synchronization with external sources of information - models, architectures, ontologies, requirements, processes, etc, - anything that helps define the enterprise and how it operates.

Architects use "OsEra View" tools to manage, design and integrate the information from the source material, creating a unified view of the enterprise from business requirements to system interfaces. Views may be provided by "internal" or external tools, integrating the specification and modeling environment. The architecture design and integration process is assisted with smart, ontology driven infrencing engines.

OsEra architectures are not an end, but a means to an end – to improve the enterprise. Using MDA techniques, the architectures will help produce acquisition specifications, documentation, system specifications, DBMS systems, ontologies, component implementation and workflow systems.

Requirements

The following are some of the requirements that have been identified for the metadata infrastructure and the inference capabilities used in that infrastructure.

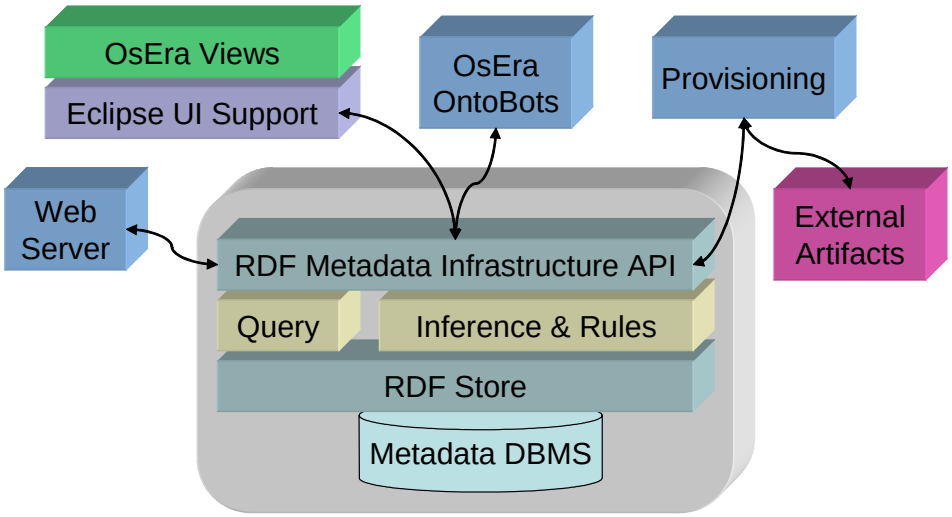
- Our meta-meta model is designed to express concepts of languages, it is "meta recursive" in that it is self-expressive so that we can express semantics of the concepts we are describing (E.G. create the concept of "role" and instantiate it). Note that this conflicts with a basic OWL-DL restriction.
- Extensibility; we can add concepts and inference that corresponds to the semantics we are describing. (E.G. adding the concept of sequence - this happens after that) We need to capture a wide range of the semantics used for architectures. We can not limit the concepts represented to those that can be reasoned over, but we should be able to inference across those concepts that can support it.
- It must work in "real time" as we load and work with models.
- Inferencing needs to assist the logical process and validation of architectures, not prove theories.
- Architectures embody specifications, the use of ontologies for specification is somewhat different than knowledge discovery or describing human knowlage in general. It is more reasonable to say that some statement violates the specification that to make many of the assumptions that reasoners currently make – such as that 2 individuals are in fact the same individual.
- OsEra will contain multiple architectures that make statements about the same thing – these will be the point of integration. These architectures, some of which will be existing, will not necessarily be consistent with each other. We need to be able to embrace different points of view, conflicting definitions and a concept of context. Ontology tools should help in finding these points of integration and resolving conflicts.
- Architectures change over time and the ability to understand that change is crucial. We need to understand the provenance of information and retain a history of change with the provenance of those changes.
- Concepts of context and provenance should be able to disambiguate multiple points of view and change over time.
- Architectures should be able to be expressed and related at multiple levels of abstraction, from business requirements through technology implementations. Where possible automated provisioning shall be used. Where automation is not practical, traceability should be provided between the levels of abstraction and different views.

- Whatever abstractions OsEra deals with can not be a “closed set”, it needs the ability to be expanded as new models, idioms and languages are integrated. OsEra can not limit the expressivity of the languages it integrates.

- OsEra will not be one thing in one place, there will most likely be no central point of control for architectures. The set of architectures must be provided by a distributed and federated system of systems.

- OsEra will provide a unified model of the enterprise but this is not monolithic. The model (or ontology) of the enterprise must be accessible with a set of views, each designed for its purpose and audience. However, these views are not independent – they must be views of the same underlying model. Changes made in one view must be consistently propagated to other views through the core OsEra view (Semantic core).

Metadata Infrastructure Architecture



The OsEra Metadata infrastructure will contain the following technology components;

- An RDF Store capable of managing sets of enterprise scale architecture with concurrent distributed development.
- A DBMS to hold the architectures and their metadata
- An RDF query processor in support of access to the RDF store, support inferencing, provisioning and the user interface layer.
- One or more inference and rules engines that help integrate, expand and transform the metadata in support of the other layers.
- An API into the metadata supported by query and inferencing. The API will be able to access an extended view of the underlying models with the semantics and

implications of that model extended by the inferencing support. (For example, a subclass will show the features and constraints of its superclasses).

- Transformation; We expect query, rules and inference to support transformation. It is unclear if there is additional support required for transformation, something along the lines of a RDF-QVT. Our working assumption is that between query and rules there will be sufficient capability but this requires more investigation.
- The structure of OsEra and the models must allow for separation of concerns across multiple dimensions. Of utmost importance is the separation between business concerns and technology concerns. Architectures should be as free from technical implications as possible, allowing greater freedom in the selection and integration of multiple architectures using the MDA design pattern.

The Metadata infrastructure will support;

- OsEra "OntoBots" that will use ontologies and inference to help define, manage and integrate architectures. OntoBots are pluggable capabilities that assist the architecture and implementation processes using advanced reasoning.
- External tools and information through provisioning
- A model driven provisioning layer that transforms external systems, architectures and information into and out of the metadata infrastructure.
- A web server to manage the access to the architecture repository from browsers as well as external metadata clients using web services
- The eclipse UI layer supporting views of the integrated OsEra architectures.

Reference Ontologies

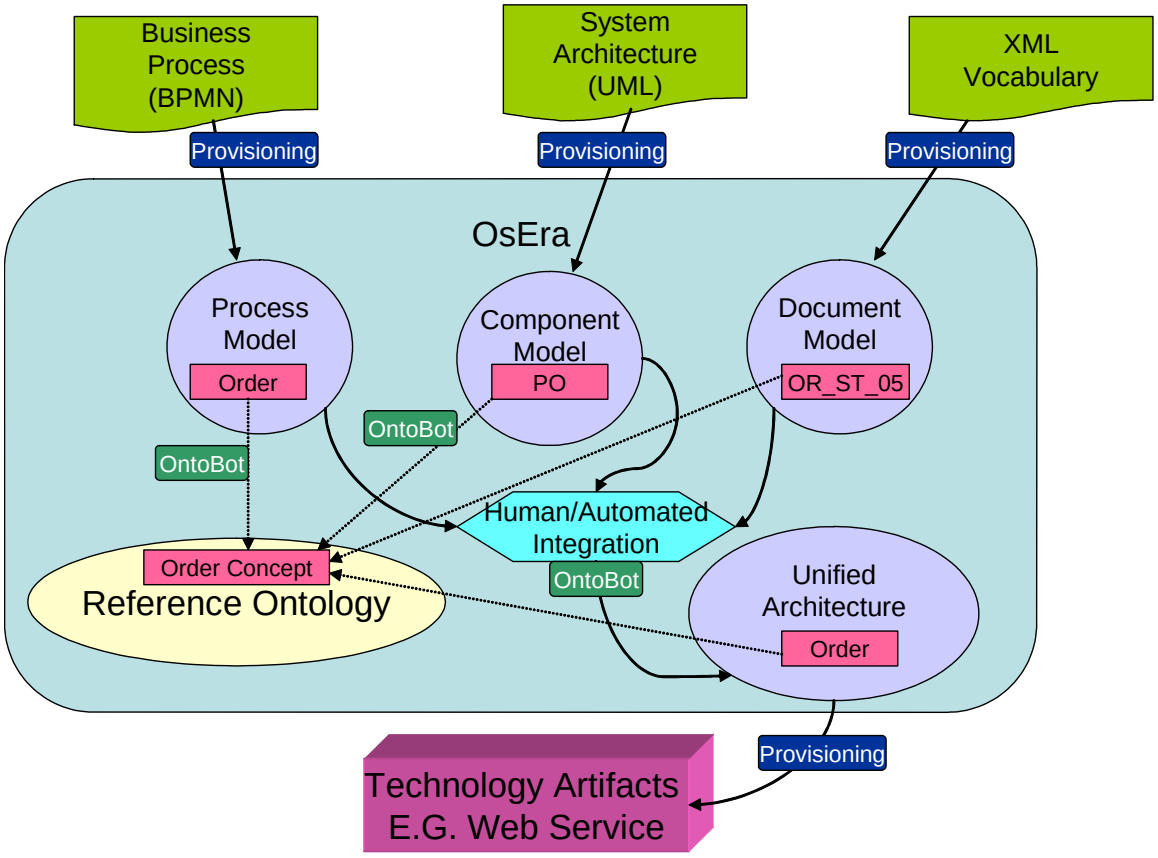
Reference ontologies assist in architectural alignment. Consider that an organization wants an integrated architecture for order management, as part of the analysis they have imported three architecture models into OsEra. One is a system design model for an order management system. One is a business process for order management and the last is an XML vocabulary for orders. All of these are clearly related and relevant. But, of course, they all use different terms when they are "talking" about the same things, express different parts of the problem and are not completely consistent.

Just by importing these models into OsEra there is some advantage in that they can be looked at in the same way with the same tools – OsEra can translate them into the desired views. The information model of the system architecture can be compared with the XML vocabulary. But we would really like more.

Within OsEra there will be reference ontologies for common concepts. As part of unification, the imported architectures are "tagged" with their relationship to these common concepts. For example, an "order" and a "PO" may be the same thing but may have different names. OsEra provides a way to match the terms and concepts of imported architectures with existing concepts, and to create new reference concepts as more architectures are created. These reference architectures are shard and become part of the OsEra environment. Tagging an architecture is partially automated and partially manual.

OsEra OntoBots will suggest mappings that will be confirmed by the analyst. Concepts that can't be matched will be manually marked by the analyst. Where new concepts are uncovered another OntoBot helps the user define that concept. This process is referred to as "grounding" the architecture.

With architectures and specifications "marked" against a reference, additional OntoBots can start to guide the user into creating a merged and consistent architecture – with full traceability back to the original information. Reference ontologies are the glue between previously unrelated information. The initial reference ontologies are populated from existing source, such as WordNet and SUMO.



The above diagram shows how these separate artifacts are brought into OsEra, tagged with a common concept – such as an order, and then a unified architecture is produced. OsEra provisioning is used to import the existing architecture models as well as to generate technology and other artifacts, such as web service specifications.

OsEra provides a general architecture for semantic integration and a "pluggable architecture" for OntoBots such that as smarter OntoBots are developed with more advanced technologies (this is a rapidly advancing field) they can be plugged into OsEra without damaging the integrity of the architectures in place. In fact, multiple OntoBots could be used together, each for its best purpose.

Semantic Core

The infrastructure architecture, above discusses the technologies to manage architectures. The semantic core is for managing the semantics of these architectures. Semantic core is the reference ontology for architecture and also the "meta model" for the architectures that are represented in OsEra.

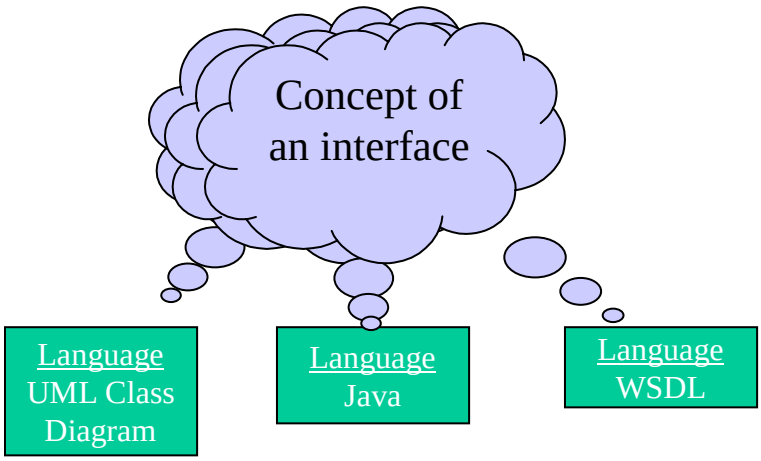
While a plethora of languages and notations have been used to specify architectures, there is a great degree of overlap in what those languages can say – their semantics. There is also inconsistency in how they are used and combined, essentially idioms of expression (For example, different architects use UML or OWL in very different ways).

The complicating factor is that different languages and idioms combine these concepts in very different ways and sometimes use very different ways to express similar concepts. In semantic core we have "sliced and diced" these concepts into smaller units we call "semantic components". Each semantic component represents a specific fact in the architecture. Some semantic components are very small and atomic while others aggregate concepts into common packages and patterns.

The source languages have been analyzed to discover the underlying semantics and define these as semantic components, wherever possible having just one way to express a particular idea. The set of semantic components is the semantic core. No one graphical or textual language may have all the semantic components; languages tend to express a subset of these concepts for particular purposes. A particular syntax or file format is mapped to its semantic core representation when it is imported into or exported from OsEra.

When an architecture model is referenced in OsEra it is mapped to a set of instances in the RDF store that are instances of the semantic core concepts.

Example;



For example, the computer languages UML, Java and WSDL all have a concept of an interface, but with some "extra sauce" attached to each. The semantic core provides a single concept of an interface, and separates the special sauce of each language.

Mapping architectures to the semantic core as a "hub" means that each language or idiom only needs to be mapped once to be integrated with all other semantic core enabled

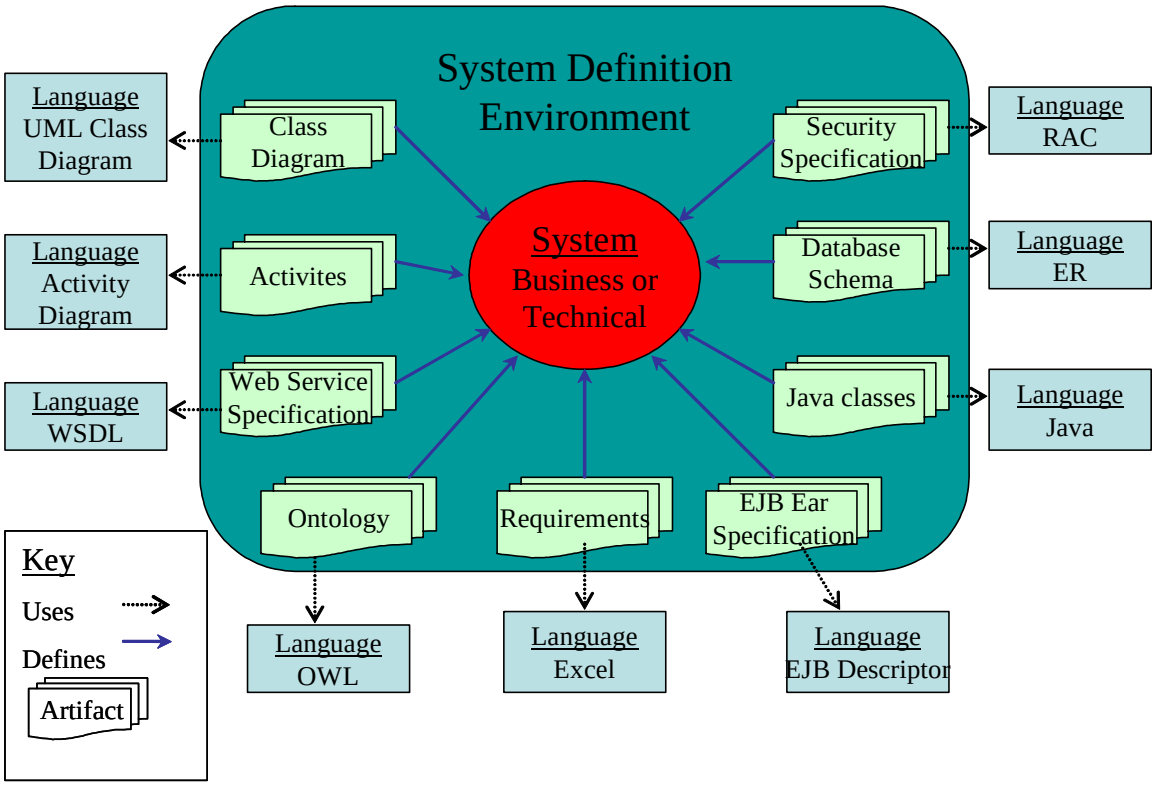
languages. It also means that architectures can be built that integrate the information found in multiple source models in multiple languages, something that could not be done with a point to point mapping between languages. The whole is more than the sum of its parts as patterns of information emerge.

As new languages and idioms are integrated into OsEra,new semantic components are created to account for any new concepts introduced. In this way OsEra grows in its capability, it is not a fixed model that everything must adapt to – OsEra adapts to the architectures and languages it represents and integrates.

In the initial phases semantic core components will be represented with RDFS and a subset of OWL, as semantic core is more developed it will be defined in it's self but still utilize RDF and OWL technologies.

This approach to metadata management has several advantages;

- Integration of information from diverse sources
- Translation between languages and views
- Semantic integration of architectures
- Support for OntoBots and provisioning from a single source or target
- Traceability between the source, derivation and use of information
- Independence from specific forms of diagrams, language syntaxes or technologies.



Consider the diagram from the start of this document. We may now consider all of the parts of the system specification *views on a common enterprise model* instead of unrelated documents. The common enterprise model is more than the sum of its parts and the information in one view may leverage the information in another view.

Concepts in the semantic core

The semantic core is designed to be extensible, but the initial core has a specific range of concepts to be expressed, those that are the most common in government architecture. The areas of concern are;

- Requirements and requirement satisfaction
- Information and data modeling (E.G. UML class modeling, Entity relational)
- Business processes (E.G. UML activity diagrams, BPMN)
- Collaboration (E.G. OMG Edoc, EBXML)
- Organizational Structure
- Value Chains
- Security Profiles
- Service interfaces (E.G. Web services)
- Federal Enterprise Architecture
- Source and pedigree of information

- Ontologies (E.G. OWL)
- Description of the core it's self (E.G. RDF Schema)
- Technology Mapping (E.G. MOF-QVT)

Defining semantic components

The semantic core, as a set of related semantic components will be defined in a smaller set of "meta meta concepts". The primary meta meta language used for defining Semantic Core (particularly in the bootstrap phase) will be RDF-Schema. RDF Schema provides the basic concepts we need and a strong extension capability with few limitations. Additional concepts, as required, will be utilized from OWL.

It is our expectation that this definition will utilize these core RDF-Schema concepts combined with a rule based reasoner to support the inference capability required for the metadata infrastructure. One reason for this approach is to allow the "meta circularity" and extensibility that is at the core of the semantic core, which is not a capability of OWL-DL.

Discussion of rule based approach

An approach to reasoners that seems important here is; OWL-DLP (http://logic.aifb.uni-karlsruhe.de/) seems to be the formalism of the horn clause approach. This is defined is the intersection of DL and horn clause reasoners – here is a formal definition:

http://www.cs.man.ac.uk/~horrocks/Publications/download/2003/p117-grosof.pdf The rule based (horn clause) approach seems to imply some capability to express new inference rules as well as a relaxation on the restriction against meta-recursion. Also keep in mind that we will certainly be expressing concepts not in OWL, very specifically time and context based semantics. (Describing processes without describing "change" is a bit limiting).

As defined, OWL-DLP does not relax the meta-restriction as it is a pure subset of OWL-DL. However, the reasoners that use the horn clause approach generally relax that restriction – it is a well known requirement. This seems to be what is used in SWRL (http://www.daml.org/rules/proposal/). So it looks like OWL-DLP+SWRL provides for fast inference, extensibility and (possibly) a relaxation of the meta restriction. It doesn't look like any of the other OWL-DLP restrictions are a problem.

Another pragmatic consideration is TURNING OFF some inference. From the perspective of a specification, the way these reasoners join instances to satisfy constraints seems like the wrong answer 90% of the time. It is great to be able to say this and that are the same thing, it is wrong to assume so just because some fact 100 yards away implies it (suggests it – fine, but that is not what happens). We would really like something more like an "error message" when constraints are violated instead of deciding the president is a rock (no comments on similarity, please).

So what I suspect is that the kind of inferencing we need is developing along this DLP/SWRL line, more so than the DL reasoners, like Pellet. Also, that while not yet standard, the tools implementing this allow for both extensibility (by defining new rules) as well as meta-circularity. A specific instance of this is RDF Gateway (not sure about Jena or Sesame – more investigation is required) that explicitly supports these capabilities.

From a pragmatic point of view, it seems very clear that you could express the kind of rules we need – here is the part of RDF gateway that talks about it: http://www.intellidimension.com/pages/rdfgateway/dev-guide/db/rules.rsp. To get things going

(bootstrap) it may make sense to use RDF-Gateway even if it is not open source, but one of the Java open source platforms would be great if they can support it.

Another potential approach for the meta-circular thing is to use the class/instance pattern. But, it would be better to do that in an export mapping so that our core semantic didn't get polluted.

Finally, another approach is to just use RDFS inferencing and add our own rules for what we need (Such as cardinality) – which may not be that difficult.

So the point of this discussion is to get input on identifying the inference infrastructure we can use that satisfies the above meta-requirements. This is what we would use for the semantic core inside OsEra and probably for transformations. It is not necessarily what would be used by other tools that may process ontologies in OsEra for other purposes (E.G. identifying conflicts or potential unifications) – if those tools had other requirements (Such as OWL-DL) we would have to export a suitable view (just like generating code for a runtime engine).

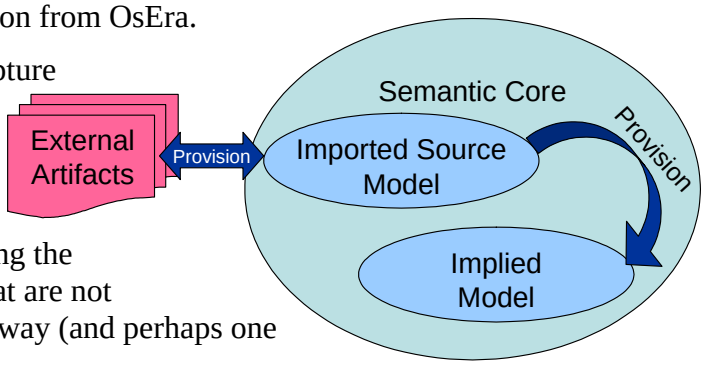
Mapping into and out of OsEra & Semantic Core

Integrating with external models and artifacts is done for a variety of reasons and with various capabilities. The following are the common use cases;

Mapping use cases

Reverse engineering

Capturing information from an external source (usually a technology) is a one-way or one time activity. Reverse engineering involves applying patterns to external information to discover the model underneath. The source of the information is captured but there is no expectation of updating the source information from OsEra.



The first step in reverse engineering is to capture the concepts in the source artifacts as a model, this may involve some user interaction. The second step is to apply patterns to produce an "implied" higher level model from the source model. Producing the implied model uses patterns and huristics that are not nessisaraly reversable, thus making it a one-way (and perhaps one time) operation.

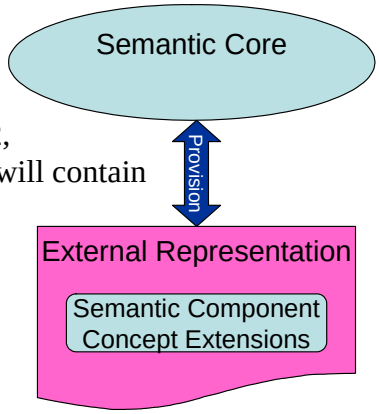
Since reverse engineering uses patterns to achieve the concept mapping and there are choices in such patterns, reverse engineering is "parameterized" – there are ways to specify the options for the mapping. Reverse engineering also may involve user interaction, choices and wizards.

External views, Round-trip modeling

Keeping an external view of an architecture in sync with the OsEra repository. Such an external view may either be a full or partial representation of the unified architecture.

Fully mapped languages

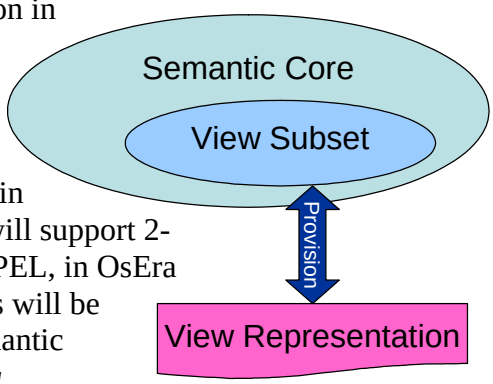
A select set of languages will be able to represent the full extent of the information in the semantic core. Initially only RDF/RDFS will have such a representation. In that the semantic core is an open set of semantic components, any fully mapped languages will require extensive extension capability. Candidates include UML-2, OWL-Full and some formal languages. A fully mapped language will contain some of the same concepts as are represented in the semantic core – these will be mapped directly. Other concepts that do not exist directly will utilize the languages extension capability. A fully mapped language is equivelent in expressability to the semantic core.



Example; The semantic core will certainly contain the concept of an "activity" (in the sense of UML-2 activity diagrams). Such a concept does not exist in OWL (Even OWL full). The concept of an activity will have to be defined in OWL as subtype of owl:class. We will then map the activity semantic component to an owl: activity.

Subset Languages (Views)

Most mappings will utilize only a subset of the information in OsEra and will map to existing models, profiles or ontologies as domain specific languages. For example, BPEL (Business Process Specification Language) contains a well developed set of concepts for activity modeling. An activity modeling view of the information in OsEra will be mapped to BPEL.. This type of mapping will support 2-way integration, changes to the model may be made in BPEL, in OsEra or in another view that supports activities. These changes will be merged back into a model in OsEra as defined by the semantic core. *A view is a subset of the semantic core mapped to a representation in another form.*

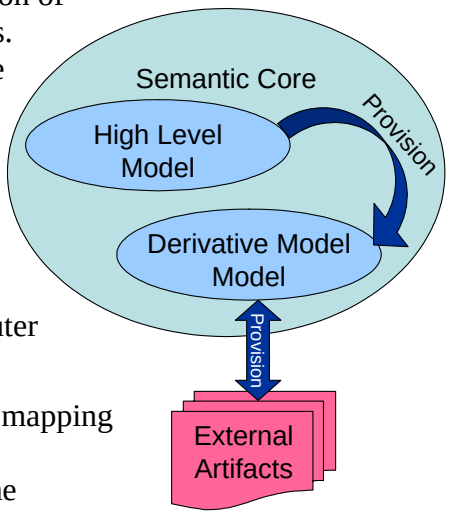


Another, similar, example is OWL-S, OWL-S provides for an ontology of a web service. An OWL-S mapping will use just the concepts defined in OWL-S and be faithful to that view, but users of that view will not have direct access to the other information in OsEra – for that they would need to look at another view.

Some "native" views will be simply user interfaces on the OsEra models, implemented in the OsEra environment.

Derivation & Artifact Generation

Artifact generation is the production of documents, specifications, code or implementations at a lower level of abstraction – a combination of producing a derived model and provisioning external artifacts. Artifact generation is used to specify or implement how some concept is to be achieved based on specific and selectable patterns. For example, the concept of a business interface may generate various Java and J2EE artifacts to implement that business interface. Due to the information loss on generation and since various and complex patterns are used that are not necessarily reversible, artifact generation is one-way. A common example of this pattern is the typical computer language compiler.



Since artifact generation uses patterns to achieve the concept mapping and there are choices in such patterns, artifact generation is "parameterized" – there are ways to specify the options for the mapping.

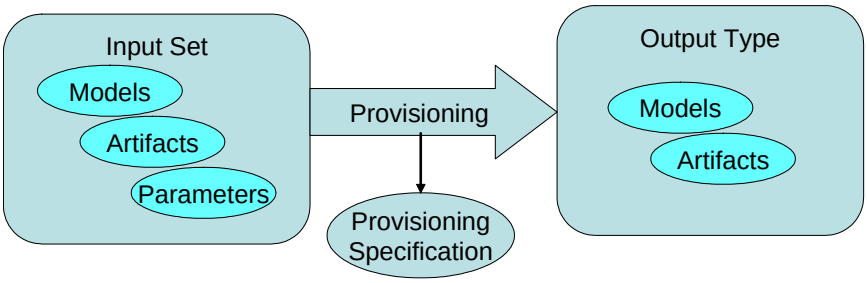
Artifact generation includes "code generation", production of specifications (E.G. for RFPs), Documentation, etc. This is the most discuss part of utilizing MDA but is in fact, only one of its value propositions.

Mapping Concepts

Round trip modeling & preservation of concept identity

External models and views that are synchronized with OsEra support 2-way integration (Round trip engineering). To support round-trip engineering and the integrity of models the preservation of a concepts "identity" is crucial. When a concept (instance of a semantic component) is exported it will be exported with a URI of the concept in OsEra. When information is imported from the external view this URI is retained such that it is clear what concepts are being changed, added or removed. Concept identity is a requirement for a 2-way mapping and may use standard or ad-hoc mechanisms of the target language.

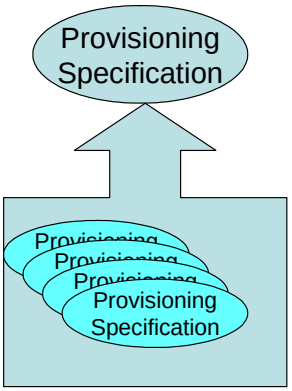
Provisioning Specification



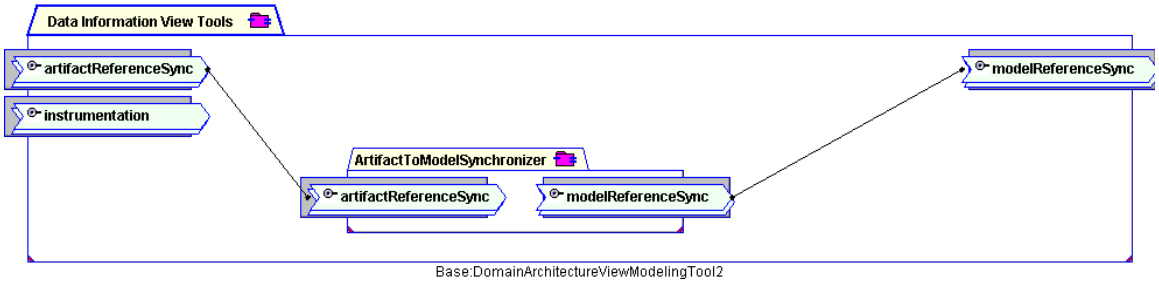
To map into or out of semantic core, or to provision from one layer of abstraction to another within semantic core, requires a provisioning specification for that mapping. A mapping specification must exist or be derivable for the specific set of input information and the desired set of output information.

The provisioning specification is dynamically bound based on the information in the input set and the desired output type.

The provision specification is not monolithic but is it's self composed of sub-specifications. For example, at the "top level" there may be a "Semantic core to WSDL" mapping, but this would be composed of very detailed sub-components such as "information flow to operation".

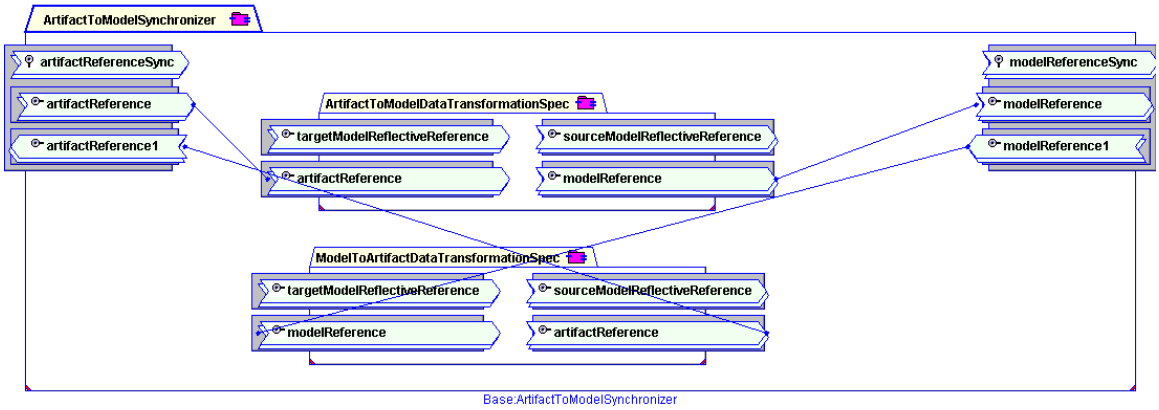


In the detailed architecture this provisioning pattern is evident in multiple places. For example



This provisioning component is responsible for the integration between an information modeling view and semantic core.

Inside this component there are provisioning specifications for each "direction"



Each of which may be composed of multiple sub-components.

Detailed specification of provisioning

At this time, specification of provisioning uses a combination of model and technology-specific forms, such as XSLT and ANT. Ultimately provisioning specification will be fully model based, perhaps using formal methods.