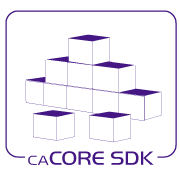
Next Generation Service Development Kit



Version: 0.1

Last Modified: 8/30/2010

Team : caCORE SDK

Client : National Cancer Institute -   
 Center for Bioinformatics and Information Technology,

National Institutes of Health,

US Department of Health and Human Services

Document History

Revision History

| **Version Number** | **Revision Date** | **Author** | **Summary of Changes** |
| --- | --- | --- | --- |
| 0.1 | 06/30/2010 | Bediako George | Initial Draft |
| 0.2 | 07/23/2010 | Bediako George | Incorporated more information regarding Open Health Tools, SDK comparison chart, and Open ID. Updated references and XHTML representation diagram. Added more information in footnotes. Fixed typos and grammatical mistakes. Incorporated changes and additions recommended by Sichen Liu. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Review

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Team/Role** | **Version** | **Date Reviewed** | **Reviewer Comments** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Related Documents

More information can be found in the following related documents:

|  |
| --- |
| **Document Name** |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

Table of Contents

1. Executive Summary 5

2. Introduction 6

3. Interoperability Concepts 7

3.1 Working Interoperability 8

3.2 Syntactic Interoperability 8

3.3 Semantic Interoperability 8

4. Modeling 9

4.1 Modeling Meaning 9

4.2 Modeling Object Persistence 10

4.3 Modeling Object Representation 12

4.4 Modeling Visual Presentation 15

4.5 Modeling Application Security 17

4.6 Modeling Documentation 17

5. Representing Models 18

6. Artifact Generation 20

6.1 Generation Approach 20

6.2 Generation Targets 21

6.3 Multiple Language Support 24

6.4 Presentation 25

6.5 Concepts 25

6.6 Documentation and Artifacts 25

6.7 Validation 26

6.8 Performance 26

7. Services 26

7.1 SOAP based Web Services 27

7.2 Web Service Discovery via WSDL-S 27

7.3 RESTful Web Services via HTTP 28

7.4 RESTful Service Discovery via HTTP 29

7.5 SOAP over REST 30

8. SDK Development Approach 30

8.1 Reuse existing infrastructure 30

8.2 Build Our Own by Copying Others 30

8.3 Candidate SDK Intermediate Form - EMF Ecore 31

8.4 Challenges 32

8.5 Solution 33

9. Enterprise Compliance 33

10. Open Health Tools 34

11. Conclusion 35

12. References 35

13. Appendix 36

13.1 SDK v. SDK Feature Matrix 36

# Executive Summary

The caCORE tools and APIs developed by the National Cancer Institute Center for Bioinformatics and Information Technology (NCI CBIIT) provide the building blocks for development of interoperable information management systems. As the caBIG® community continues to expand along with evolution of technologies, standards and policies – the Software Development Kit (SDK) team examines the need for next generation caCORE Infrastructure tools and processes to enable enterprise-grade and standards based syntactically and semantically interoperable applications in a federated collaborative environment.

The current incantation of SDK has provided excellent support for community developers trying to meet this goal. There is still room for improvement of the current incantation of the SDK.

Using popular open source frameworks as a template, along with a fuller understanding of the types of development issues that come with the production of a popular application kit, the beginnings of a new Service Development Kit[[1]](#footnote-1) (SDK) is proposed.

The goal for the next generation SDK is to have better working interoperability support for multiple model formats as input, and better working interoperability support for the multiple artifact targets as output. To achieve this, an architecture centered about the creation of an intermediate form has been offered. An intermediate form is a meta model format that is capable of serving as a common and central representation format amongst other meta model formats. SDK has identified the Eclipse Modeling framework’s Ecore as one such intermediate form.

The new SDK will generate documentation and testing logic as well as code. This will enhance developer productivity, and meet SAIF’s ECCF requirements for working interoperability. The SDK team as highlighted several new design features that will support this goal. SDK will boast dynamic artifact generators that will gracefully support the generation of multiple language platform artifacts that will work efficiently not only on the web, but on the laptop, tablet, or the mobile phone. SDK will also embrace concept aware data representations that will automatically and explicitly identify the concepts associated with every element of data and relationships served by SDK generated RESTful and SOAP based services.

Finally, the new SDK will support usage via the command line, via service invocation, and via integrated development environment. Developer’s will be able to execute build scripts, call build services, or use an Eclipse based editor to configure and execute the new SDK generated service.

# Introduction

The Software Development Kit (SDK) created by the caCORE team is a code generation utility designed with the principal purpose of exposing information stored in persistent systems to human beings and computing systems.  In order to enable these capabilities, the SDK would read information from a UML diagram then generate the required Java artifacts to expose data from relational database stores via Spring Remoting based web services.  
  
At the time of SDK’s inception, this approach was considered cutting edge.  Model driven design (MDD) was becoming established in mainstream development circles.  Java, the target language of choice, enjoyed dominance as an enterprise platform.   Hibernate was the leading object relational management library in the Java world, and Spring Remoting represented an easy to deploy and stable service protocol for enterprise worthy web services.   The direction chosen for development of SDK was clearly influenced by the technology environment of the time.  
  
This choice of action resulted in great success.  Today, the latest version of SDK is a robust application, capable of reliably meeting the aforementioned goals.  Many developers rely on SDK to stand up their mission critical applications.  
  
This success has not been complete.  The current design of SDK has been limited by a philosophy of reuse of Java specific tools for enterprise computing.  The result has been the creation of a product Java developers can use to create data services.  Notably, other development environments have been left out.  
  
For Java developers, the downside of using popular Java frameworks to accomplish the creation of web services is that library collisions have become a real issue.  Trying to manage all of the different changes that occur amongst all the packages used to implement persistence and web services has proven to be a challenge.  
  
This organization’s stated goal to promote working interoperability makes it clear that this cannot be completely met using a Java based product.  .Net, Python, php, Ruby, JavaScript, and other language developers are not able to benefit from the SDK, even though the use of these dynamic languages for enterprise development is increasing at a rapid pace.  In order for the SDK to stay relevant it will have to be able to address the needs of these developers.  
  
It should be clear that the current incantation of SDK would prove to be inadequate in the not too distant future from both an architectural and a design perspective.  The following lends some perspective to this premise:

* Our increase in knowledge in the proper creation of open and scalable web services indicates that having SOAP based web services is not enough.  There are other considerations that support semantic interoperability that are not adequately addressed by WSDL alone.
* Our better understanding of knowledge representation and alternate knowledge stores indicates that supporting just UML and relational databases is insufficient.
* Our desire to support the automatic discovery and use of many disparate sources of information by many disparate types of systems strongly suggests that supporting Java based systems alone would leave out a large and still growing development community.

A language toolkit that supports a single platform is not enough.  The new SDK should not be a product that serves one slice of the computing environment.  Instead, it should become a framework capable of supporting a multitude of platform environments.   This would speed adoption, and will encourage the growth of the open source development pool to one beyond the size of the current SDK team. To achieve this goal, the right seeds must first be planted.  A new SDK would require new direction in design:

* It would need to be modular and extensible.
* It should support the generation of multiple artifacts for multiple application layers for multiple given platforms. Developers should be able to gracefully add new functionality generators as they see fit.
* It should not be tied down to simply one knowledge representation format, but instead be based on a format that for all practical purposes can represent the widely accepted foundational entities that support model driven architecture.

Achieving these bullet points is possible.  Much work has been accomplished in the last 10 years with regards to advanced knowledge representations, scalable and interoperable services, informative data representations, and model driven engineering.  In fact, it is apparent that these areas are starting to converge towards a universal standard of computing.  This standard is not yet formally defined, but the convergence of these design philosophies indicates that it emergence is imminent.  
  
Much of the work accomplished by the Object Management Group (OMG) and the World Wide Web Consortium (W3C) reflects this sentiment.  The next generation Service Development Kit (SDK) will leverage these expert thoughts surrounding the creation of open multi platform computing and services to create a new SDK that embodies much of the new ideas mentioned in the previous chapters.

# Interoperability Concepts

To fully understand the motivations behind the recommendations in this document, it would be helpful to understand the notions considered while researching future Software Development Kit (SDK) capabilities.  The following paragraphs identify these major concepts.

## Working Interoperability

From the HL7 SAIF Book document, working interoperability is described as:

*The collection of structures, processes, and components that support Computable Semantic Interoperability (CSI) between two parties (“trading partners”) who are interacting (for example, exchanging information, coordinating behavior) to achieve one or more business goals.* ***Interoperability****, in this context, is further defined to be the* ***deterministic*** *exchange of data or information in a manner that* ***preserves shared meaning****.*

In other words, working interoperability refers to the ability for a computing system to safely receive and completely understand information from another computing system.  As a matter of course working interoperability depends on these systems ability to support both syntactic interoperability and semantic interoperability. Syntactic interoperability involves the ability to properly parse structured or semi-structured information as dictated by a common data format agreed upon by two computing systems.  Successful semantic interoperability depends on syntactic interoperability being present, and it refers to the ability of a computing system to construe the intended meaning of a parsed stream of data sent by another computing system.  
  
The goal of caCORE is the provide development tools that facilitate the creation of computing services.  These services can only be effectively used by other computing systems if working interoperability is supported.  It is imperative that services created by SDK support both semantic and syntactic interoperability.

## Syntactic Interoperability

Syntactic Interoperability relies on the processing of information syntax.  Syntax refers to the essential elements of data representation.  An analogy to the syntax learned in grammar school is appropriate.  “Data elements” could be thought of as words in a sentence; “data structures” could be thought of as sentences, and multiple related structures could be thought of as paragraphs.  Data types, data format, data length and size are all data element or word level concepts, while data element order and data element dependencies are sentence level concepts.

## Semantic Interoperability

An agreed upon meaning can be attached to any given data element.  That meaning, or concept, may be physically represented by an agreed upon designation or name, however the meaning signified by that designation is a separate and abstract representation of that thought or concept.  As computing systems process a stream of data elements represented by separate designations, successful computation of that stream of data elements is only possible if the meaning construed by the sending computing system is the same as the meaning interpreted by the receiving computing system.  It also is possible for a computing system to successfully parse a stream of incoming data, but not be able to apply any meaning to the designations it has parsed.  Complete understanding of a message is only possible when syntactic and semantic processing are both successful.

# Modeling

Generally speaking a model is a construct that is used to represent something else.  In software development, a model is a construct that is used to represent a collection of ideas. Software models include but are not limited to conceptual, logical, and physical models.  Conceptual models are used to model business constructs, logical models provide more detail surrounding those business constructs and also represent the business relationships amongst those constructs.  Physical models provide a concrete representation that is usually specific to a particular software platform implementation.

The use of model driven engineering, or MDE, has become popular in the realm of data persistence and storage; in truth many other areas of software development may benefit from a modeling based approach.  Indeed, almost all parts of software development could be modeled.  Models have been found useful for describing:

* Meaning
* Object persistence
* Object representations
* Visual presentation
* System security
* Documentation

Any future implementation of SDK should provide modeling and artifact generation for all of these areas.  The following chapters give more detailed descriptions of these salient modeling concepts.

## Modeling Meaning

In order to effectively model the meaning behind an enterprise application, a model framework should support several basic data construct concepts. The diagram below illustrates how these modeling concepts are related:

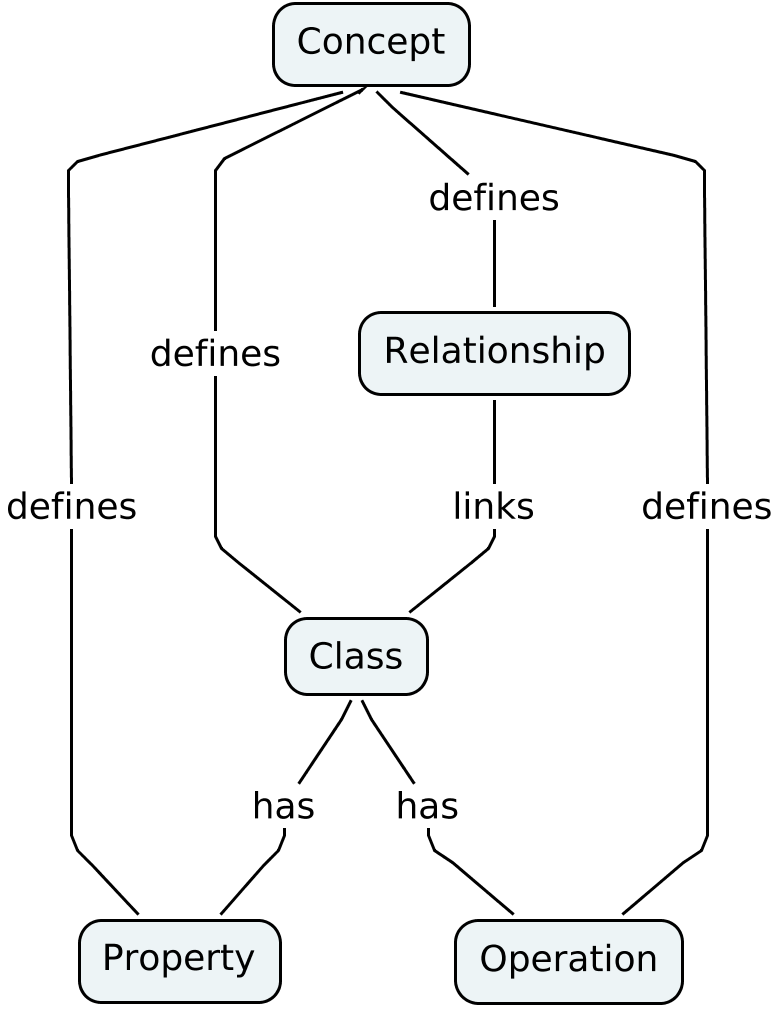


Figure : Fundamental parts of meaning model representations.

For the purposes of this exercise the SDK team has identified the following set of model entities as the group of concepts necessary and sufficient to represent meaning:

* **Concept** - A concept is a distinct quantum of meaning.
* **Property** - A property is a container for a value.  Properties are associated with classes.  Properties have meaning and as such are attached to a set of concepts.
* **Operation** - An operation is a representation of an action that can be applied to an ordered list of values or instance of a class.  Operations have meaning and as such are attached to a set of concepts.
* **Class** -A class is a collection of properties and operations. Classes are usually used to group properties and operations that are related in some manner.  Classes have meaning and as such are attached to a set of concepts.
* **Relationship** - A relationship describes a meaningful connection between two classes.  Relationships have meaning and as such are attached to a set of concepts.

Any modeling framework that hopes to be able to support the goals on semantic interoperability will have to be able to effectively represent the aforementioned constructs.

## Modeling Object Persistence

An object is a single instance of a class.  In other words, objects are used to store information that is peculiar to an entity that belongs to one or more classes.  Often information in objects must be stored in a safe place for later retrieval.  There are rules that define the mechanisms under which an object is stored.  Any modeling framework will have to be able to represent these storage rules.  
  
The following list identifies the types of storage rules and concepts that must be representative in a persistence model.

* **Object** - an object is a single instance of a class.
* **Attribute** - an attribute is an instance of a property, and it belongs to an object.
* **Data type** – A data type is a classification of a value.  Data types have meaning and as such are attached to a set of concepts.
* **Property constraint** – A property constraint is a rule defining the set of allowed values of the property or its relationship to other properties.
* **Relationship constraint** – A relationship constraint is a rule defining the conditions under which a relationship between two classes must conform.
* **Unique identifier** - a unique identifier is a property whose value can be used to uniquely identify a particular instance of a class.  A unique identifier should contain enough information about an object instance so that the exact object instance can be referenced at any time.
* **Mutable operation** - A mutable operation describes an operation that is expected to change an object’s value.
* **Safe operation** - A safe operation describes an operation that will not change an object’s value.  Retrieval of an object from its persistent store is an example of a safe operation.
* **Idempotent operation** - An idempotent operation describes an operation that returns a result based on a single invocation on a given list of input, any subsequent invocations, for that same set of input, will not result in a different result.
* **Immutable property**- An immutable property is not allowed to change its value.  An example of an immutable property is the unique identifier on that property.
* **Role** - a role is a classification of the part an individual or a system plays when executing a particular operation on an object.
* **Object change capture** - object change capture refers to the practice of tracking the history of augmentations that occurred on a particular object.  Information that is included in a captured change includes the object’s unique identifier, the values of object’s properties at the time of the change, the object’s relationships at the time of the change, the identifier referencing the entity responsible for the change, the name of the operation used to effect that change, and the time at which the change operation was executed.
* **Index** - an index is a data construct that facilitates the fast retrieval of an object from persistent store.
* **Cache entry** - a cache entry is a data construct that facilitates the fast retrieval of an object from a service or operation.
* **Collection -** bunches of objects may be stored in a container called a collection.  A collection may have a unique identifier that references that container.  Or it may be identified as the result of a function operating on an ordered list of input.  Functions of this nature are typically called queries.

## Modeling Object Representation

An instance of an object can be represented in many ways.  The following lists several representation formats that are used today.  Each representation format has it benefits and drawbacks:

* **Extensible Markup Language (XML)** - a text based data representation format that can flexibly model meaning constructs.  Since it is text based, it enjoys wide language and platform acceptance.  There are several object representation formats that are based on XML.
  + **Relationship Representations**[[2]](#footnote-2) -  Relationships between domain objects shall be modeled as links.  The W3C defines links as (X)HTML constructs that point to related representations.  Both the anchor (<a></a>) and the link (<link></link>) element tags support the “rel” and  “rev” attributes; “rel” describes the relationship of the representation to the link target, whereas “rev” describes the reverse of that relationship.  These attributes can be used to express relationships such as “associations” in UML, “objectProperties” in OWL, and “foreign keys” in DDL.  For instance, if Malik is Maliaka’s brother and Maliaka is Malik’s sister, Maliaka could create a link on her online profile page that could point to her brother’s online profile page in a semantically rich manner using the (X)HTML link construct[[3]](#footnote-3). There are also canned values for both “rel” and “rev” that give context and describes the nature of the link or relationship[[4]](#footnote-4), including “alternate”, “edit”, and “describedby”.  Of particular interest is “describedby” as it provides a standard and formal way for SDK code generator to link a representation to a formal concept that may be stored in a terminology service.  Linking representations to formal concepts in terminology stores fully supports this organization’s goal of semantic interoperability.
  + **XML Metadata Interchange (XMI)** - XMI is an Object Management Group (OMG) standard for representing object information via XML.  XMI is widely used amongst modelers to express both class and object instance data.  It is not suited for presentation purposes without further manipulation, and it does not enjoy the added benefit of advanced computability features, such as inference and reasoning.
  + **Resource Description Framework (RDF) -** RDF is a World Wide Web Consortium (W3C) specification designed as a metadata model.  When used with RDF Schema (RDFS), RDF enjoys inference and reasoning support, however it does not lend itself to easy presentation.
  + **Web Ontology Language (OWL) -** OWL is a knowledge representation language used for creating ontologies. It has been adopted by the W3C.  OWL is well equipped to support reasoning and inference.  It does not enjoy widespread use along the lines of XMI or even RDF, although its use is growing.  It is also not suited for presentation purposes as browsers are not by default OWL aware.
  + **Resource Description Framework – in – attributes (RDFa)** - RDFa is a recommendation from the W3C for embedding metadata in (X)HTML documents via RDF triples.  RDFa enjoys many of the same benefits as RDF.   It also has the added benefit of being presentation friendly, as browsers can easily show RDFa as it appears to be (X)HTML.The implication of this is that it is possible to create one data representation format that serves the purpose of providing data as both syntactically and semantically digestible to both computing systems and human beings alike.
  + **Microformats** - microformats is a web based technique for representing object instance and metadata using (X)HTML markup.  Microformats does not support inference or reasoning, however much like RDFa it makes the embedding of semantically rich meta data in an popular presentation meduim like (X)HTML possible.
  + **Atom Syndication Format (Atom) and Atom Publishing Protocol (AtomPub)** - Atom is a text based format that supports the representation of collections of resources as consumable feeds.  Both Atom and AtomPub provide a widely used standard for representing information that can be consumed by both computer and human beings alike.  Most web browsers can effortlessly display information formatted using Atom, and computing processes can easily parse Atom feeds as they are XML based.  Atom, like (X)HTML, supports link representations and provides a means to categorize data entries.  This categorization mechanism can be made to point to formal concepts stored in a terminology store.
* **JavaScript Object Notation (JSON)** - a text based representation format that closely resembles the object syntax used within the JavaScript language.  JSON is supported by a wide variety of languages, including Java, C, C++, Ruby, Perl, and Python.  It is not as visual presentation friendly as RDFa or microformats as most (X)HTML browsers do not know how to display JSON.
* **Data Type Representations as Healthcare Data types ISO 21090** - Health Level Seven (HL7) has decided to support data types as defined within the ISO 21090 specification[[5]](#footnote-5).  A subset of these data types are already supported to some extent by the current SDK.
* **Portable Data Formats** - Portable data formats are data formats that are widely accepted by the computing industry.  Many of these formats are specified or supported by data types definitions from ISO 21090 and ISO 11404:
  + For integer, decimal, float, and double data types, SDK will default to W3C XML Schema for lexical representations and support ISO 11404 integer and real literal formats.
  + For dependent territories and country codes SDK will default to ISO 3166.
  + For currency codes SDK will default to ISO 4217.
  + For dates, times, and date-times SDK will default to RFC 3339.
  + For language of text representation SDK will default to BCP 47.
  + For time zones SDK will default to Olson Time Zone Database entries.

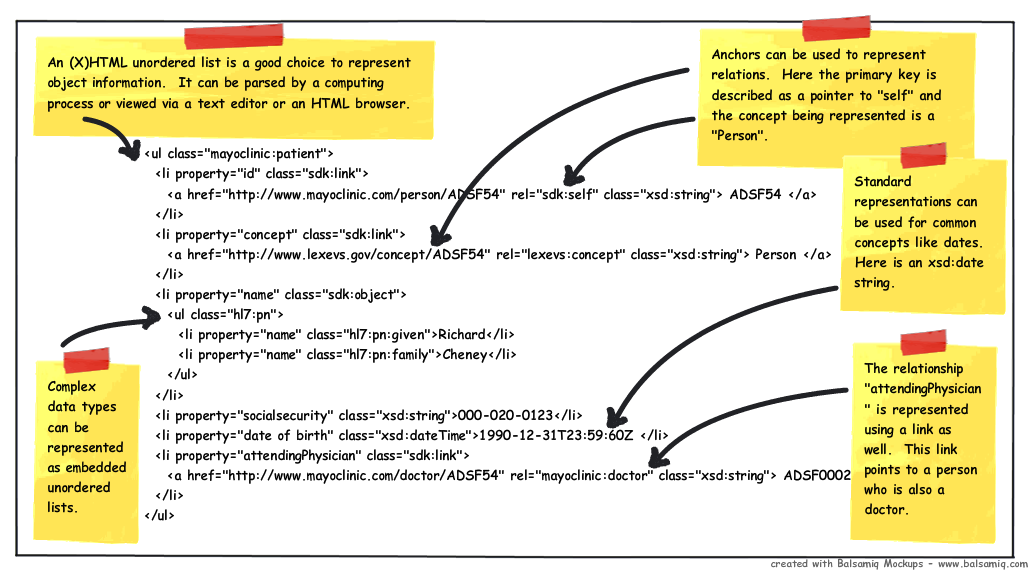


Figure : Here is an XHTML fragment representation of a person showing how object and concept information can be represented using a multitude of open standards. This representation is consumable by both human and computing process alike.

Developers can identify properties as having data types like “currency” or “country code” and as a result of that SDK should be able to assign the appropriate validation rules automatically.

All of these considerations can be combined to represent an object. In particular, the use of (X)HTML combined with XSD schema annotations can be an extremely effective means of conveying both semantic and syntactic information. Using the id and class attributes of the unordered list (<ul/>) tag in (X)HTML, SDK could represent the class, property, relationship, and concept[[6]](#footnote-6) structures that are used to model meaning in a data object. The preceding diagram illustrates how all of these considerations can come together when representing an object.

In this example, there are five models used to represent information. The first and encompassing structured representation model is that as afforded by (X)HTML. The rest are referred to in the class attribute and are in order of appearance, the Mayo Clinic data model (patient and doctor), the SDK model that describes links and objects, the HL7 complex data type model that defines data representations for common constructs encountered in the health care industry, and the XSD definitions for representing dates and strings.

Using (X)HTML in this way allows developers to reuse the same representations to help with visual presentations on the web. There are more things to consider with regards to visual presentation. These are treated in the next section.

## Modeling Visual Presentation

Information needs to be presented to human beings in formats that facilitate viewing and editing.  Human beings consume information in subjective ways, and because of this, the presentation of that information matters.  The following lists a series of concepts that are important when considering the semantic presentation of information.

* **Internationalization** - Human beings communicate using different languages. When presenting information to end users, developers need mechanisms for modeling the presentation of the language used by the graphical user interface (GUI).
* **Property groupings** - From a meaning perspective, properties are grouped in containers called classes.  When presenting the information in these properties to a human being, the class groupings may not represent the best way to gather the properties on a computer screen.  Alternative grouping strategies such as filters, facets, or composites may allow for a more friendly presentation.  These viewer friendly groups will not necessarily be used when the data items are persisted, so developers will also require the ability to map these presentation items back to the class properties they came from.
* **Property presentation order** - Sometimes the order in which properties are declared in the class is not the order that they should be displayed to the human being. For instance, it may be advantageous to change the order of presentation of a group of properties in order to address possible order based bias in the human interpretation of the information in the properties.  There needs to be a way to model the default display order of properties for presentation to the end user.
* **Display rules** - Some properties are not expected to be shown to the end user. The unique identifier or the key references for a particular object are one example.  Change data like timestamps and version information may be others.  There is a need to have a model that supports the representation of display desirability at the property level.
* **Editing rules** - Some properties are not expected to be changed by users.  The presentation model will need to support the notion of mutable and immutable properties.
* **Property level messaging** - For presentations, certain messages may need to be associated to a particular property.  These messages may be informative in nature, as they describe the meaning of a property to an end user.  They may also indicate that an error has occurred due to editing by the end user.  The presentation model should support the display of messages associated with a given property.
* **Validation rules** - Validation rules are operations that take a list of properties as input and produce a list of errors as output.  These errors indicate that property constraints have been violated.  If no violations have occurred then the returned error list is empty.  Although some validation rules are strictly related to persistence (such as the maximum allowed length of a person’s last name), others have to do with presentation and interpretation (such as the allowed format for an entered date or telephone number).  A presentation model shall support the representation of validation operations, and shall also support the attachment of groups of validation operations to a particular property.
* **Alternate labeling** - For some properties the name chosen as a designation for representing that property is not appropriate for display to an end user.  Consequently, a presentation model shall have a means for choosing an alternate designation or label for viewing purposes.
* **Text display and formatting** - Sometimes the value of a text based property needs to be displayed in a particular manner.  The way it is displayed may be affected by security considerations (such as the display of the last four digits of a social security number), or perhaps internationalization standards (such as the order of month, day, and year presentation), or even visual taste (such as the choice to truncate the long text of a description so that it better fits a table row).  A presentation model shall be able to represent these different kinds of presentation considerations.

## Modeling Application Security

Security is a broad topic that can be divided into the several distinct topics privacy, integrity, authorization, and authentication.  All are important with regards to enterprise security.  SDK will implement standard industry accepted approaches for each of these security areas.

* 1. **Privacy** - SDK will support the application of encryption algorithms such as AES, Blowfish, DES, and others.  SDK generated web services shall support TLS (or SSL) based encryption for secure network messaging.
  2. **Integrity** - SDK will support the application of digital signature algorithms such as SHA1 and MD5.
  3. **Authentication** - Authentication refers to the process of uniquely identifying a human being or a computer process.  This is done as a first step in determining access to a resource such as a computing system or even a physical structure such as a building.  Authentication is hard to model as it involves many different types of credentials.  SDK will choose open authentication standards for HTTP based services supported by W3C.  The following provides a listing of open and widely used authentication techniques that SDK can implement or support:
     + HTTP Basic Authentication
     + HTTP Digest Authentication
     + HTTP Three-legged Authentication
     + HTTP Two-legged Authentication
     + OpenId Authentication
  4. **Authorization** - Authorization refers to the process of granting operational access to an authenticated user for a given resource.  In other words, authorization could be reduced to a function that accepts an authenticated principal, an operation, and a resource.  These functions will return true if and only if authorization is granted, and return false otherwise. All authorization grants shall be recorded. Universally,

**Provided there exists a “c” where “c” is an application user credentials, “r” where “r” is the application role played by the credentialed user, “a” where “a” is the action applied by the credential user, and “d” where “d” is the domain object the credential user wishes to apply “a” on.  Let “f” be a authorization function.**

**Then,**

***f (c, r, a, d) = A***

**where “A” is the resulting generated authorization based on “f”, “c”, “r”, “a”, and “d”.**

## Modeling Documentation

Wikipedia defines a document as “a bounded physical or digital representation of a body of information designed with the capacity (and usually intent) to communicate”[[7]](#footnote-7).  This description suggests that there is a good opportunity to create models for documents.  Indeed, HL7 has already created a model for clinical documents[[8]](#footnote-8).   Modeling documentation could be thought of as a combination of modeling persistence and modeling presentation.  Many of the same considerations in those two areas apply to the modeling of documentation.

# Representing Models

After considering what can be modeled, the question of what modeling framework should be used becomes pertinent.  A model that represents other models is called a meta model.  If it is at all possible to have a single meta modeling framework that can successfully represent all of the types of models we visited in the previous chapter, that would be beneficial.  The following is a treatment on the types of meta modeling techniques or frameworks that are available:

* 1. **Language specific** - a language specific representation refers to the programming language mechanisms used to represent the modeling of meaning constructs.  These support modeling of data types, constraints, properties, property containers (object classes), concepts, and relationships. Some meta modeling frameworks are better suited than others in modeling these constructs. These language based constructs are, of course, language dependent, and as such are best suited to be deployed within their own development environment.Java's annotation constructs are a good example of this.
  2. **ISO 11179 v3** - ISO 11179 v3is a meta modeling standard supporting the modeling of data types, constraints, properties, property containers (or object classes), first class concepts, and first class relationships.
  3. **OWL 2.0. -** OWL supports modeling of data types, object types, constraints, properties, property containers (classes), and first class relationships.  Concept support could be easily added and made a first class citizen for a given ontology.
  4. **Eclipse Modeling Framework (EMF) Ecore** - Ecore supports the modeling of properties, data types, property constraints, property containers (classes), and first class relationships.  Concepts are not directly supported, but they could also be added via Ecore annotation support.  Ecore enjoys support from the Eclipse community as well as IBM.
  5. **XSD, XMI, and UML 1-2**- These support the modeling of data types, constraints, properties, property containers (classes), and weaker forms of relationships like associations or aggregations.  Relationships are not first class citizens but they can be supported if a definition of a relationship is supplied.  Concepts are not supported, but they could also be added.  In the case of UML this could be accomplished via stereotypes.  With regards to XSD, additional complex types of type “Concept” could be introduced and used to model concepts as a second class entity.
  6. **Data Definition Language (DDL) -** These support the modeling of data types, limited constraints, properties as fields, property containers as tables or views, and weaker forms of relationships as foreign keys.  Relationships are not first class citizens but they can be supported if a definition of a relationship is supplied.  This definition would likely be represented as a table.  Likewise, concept modeling is not supported, but that could be addressed in a similar fashion to modeling relationships.  Finally, DDL is specifically oriented towards representing persistence models.  Models such as presentation, representation, security, and others are not efficiently modeled using DDL.

Users of SDK will certainly use language based approaches for modeling. They will also use UML modeling techniques and model representations such as XMI.  Still others will use DDL as their sole model representation.  It is apparent that any software development framework that facilitates ease of translation from one representation format to another will attain high usability ratings.  In particular, if that framework can also address the possibilities of lost information during model conversion amongst unequally powered meta models, this would be a positive development.

For instance, given the different power of ISO 11179 v3 versus DDL, it is inevitable that some attempts to meta model using DDL constructs that are stored and modeled in ISO 11179 v3 will fail.  DDL cannot natively support the notion of a “concept” for instance.  It would be ideal to have a single representation that can be gracefully transformed back in forth amongst these representations, dealing with information loss issues along the way.  Such a model would have to be able to sufficiently support the highest and most complicated demands of meta modeling.  The diagram below illustrates this idea of a universal intermediate form below.

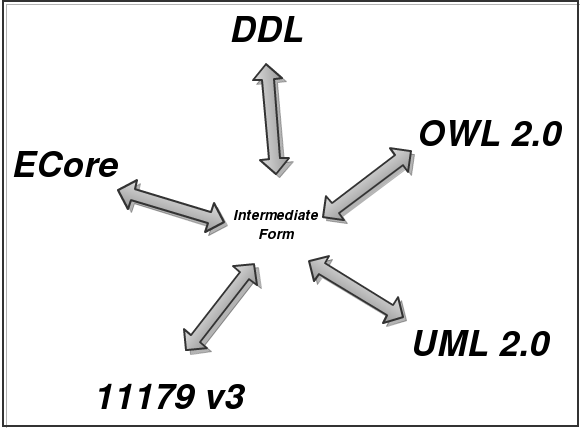


Figure : An intermediate form capable of representing the fundamental parts of modeling can serve as a universal translator for other meta models.

Of these modeling frameworks, in terms of capability considerations, ISO 11179 v3 and OWL 2.0 come closest to fitting the bill, however, for reasons of pragmatism we would not recommend using these meta models as the central modeling framework at this time.  Later in this document the SDK team will introduce a more practical approach.

# Artifact Generation

Model driven design, or MDD, is the practice of developing software from models.  MDD’s advantage over traditional means of development is that it can lead to tremendous improvements in productivity.  This is due to the possibility of transforming a given set of models, in an automated fashion, into the many artifacts that comprise an application.  The automated generation of these boilerplate artifacts saves software manufacturing time.  The artifacts generated by a program can ultimately become clearer, better written, and more maintainable than code, documentation, presentation and other constructs than those that have been written by mix of developers and business analysts of varying skill, experience, and discipline.

## Generation Approach

SDK currently employs code generation to help developers to quickly build service based applications.  Going forward, SDK should still follow this approach, but this does not mean that the current manner in which SDK generates code should be revisited.   At the very least, the following rules should be followed in order to ensure that developers get the most out of the software that is created by SDK.

* Generated code files should never have to be augmented by hand.  Reuse of generated code should be limited to inheritance, delegation, or factory methods.
* Generated artifacts should not contain any dependencies to the artifact generator that created it.  Developers should be able to reuse the generated artifacts in another environment, without having to include the libraries that were used to create the generated artifact.  In particular, generated artifacts should be created in such a way as to make reuse as easy as possible.
* Generated artifacts used to represent resource or domain objects should not contain any dependencies to the rest of the application logic.  This will ensure that the code structures that represent the domain logic can be freely shared with other developers in other related projects.  This absence of dependencies improves the opportunity for code reuse.
* Generated artifacts should not rely on the inclusion of popular development framework libraries.  Whenever possible, SDK developers should choose to fork simple open source solutions instead.  This is an easy way to avoid the difficult upgrade issues that can occur when relying on popular development frameworks.
* Generated artifacts should be stored in a separate directory structure whenever possible.  It may be the case that for the target programming environment this type of organization is not supported, however, at the very least the generated artifacts should be created in their own files.  It would be helpful to have the file name, the file contents, or the file’s directory/package structure indicate that this file was generated.
* Generated code should always automatically generate complimentary testing logic for the functional code it creates.  This could include unit tests, smoke tests, performance tests, and other forms of regression tests.

## Generation Targets

SDK currently generates Java persistence logic as well as Java specific service logic.  Persistence logic and service logic represents a traditional first cut for most code generator applications.  The future SDK artifact generator should include the following as targets:

* **Language specific class files** - source code in the target platform language that defines the resources of the domain model.
* **Language specific interfaces -** source code in the target platform language that defines the resources of the domain model.
* **Language specific data access objects** -source code in the target platform language that defines the methods for the persistent store operations create, update, read, and delete (CRUD) that are used to define, access, and mutate the resources of the domain model.
* **Language specific service logic** - source code in the target platform language that defines the service API for the operations that access resources defined in the domain model.
* **Language specific unit and regression test files** - source code in the target platform language that tests all the generated code.
* **Platform specific configuration logic** - property value definitions for the application.
* **Directory structures** - target directory structures for the generated code.
* **Uniform Resource Identifier (URI) structures** – A URI[[9]](#footnote-9) is a string that is used to identify a resource on the Internet.  A URI contains domain location, delivery protocol, network port, and resource path information.
* **Data Definition Language (DDL) scripts** - schema definition scripts for persistent stores.
* **Query Logic** - source code in the target platform language that provides support for collection, range, and relationship based inquiries of the persistent store.
* **Search Logic** - source code in the target platform language that provides support for free text natural language aware search of the persistent store.
* **Language API documentation** - documentation that provides schematics, developer guides, and usage information regarding the code generated by this application.
* **Service API documentation** - documentation that provides schematics, developer guides, resource access and availability, and information regarding the code generated by this application.  This will be key information needed to support both service discoverability and semantic interoperability.
* **Class diagrams** - a human readable diagram of the classes representing the domains within the domain model, along with the concepts associated with each class.
* **Conceptual Model Text** - a programmable list of domains within the domain model, along with the associated concepts associated with each domain.
* **Conceptual Model Diagrams** - a human readable diagram of the domains within the domain model, along with the concepts associated with each domain.
* **Logical Model Text** - a programmable list of domains, properties, and relationships, within the domain model, along with the concepts associated with each domain, property, and relationship.
* **Logical Model Diagrams** - a human readable diagram of domains, properties, and relationships, within the domain model, along with the concepts associated with each domain, property, and relationship.
* **Physical Model Text** - a programmable list of domains, properties, constraints, data types, and relationships, within the domain model, along with the concepts associated with each domain, property, and relationship.
* **Physical Model Diagrams** - a human readable diagram of domains, properties, and relationships, within the domain model, along with the concepts associated with each domain, property, and relationship.
* **Security Roles and Authorization** - a programmable map of the system user roles (human or computing process) and each roles accompanying allowed operations per resource mappings.
* **Presentation Logic** - Rudimentary data presentation logic that will allow human beings to access, edit, or otherwise manipulate domain information.
* **Caching Logic** - Logic that supports improvements of performance for services or persistent store access.
* **Indexing Logic** - Logic that supports faceted or non faceted retrieval of object instances defined by the resources defined in the domain model.
* **Resource representations** - Textual or binary representations of the object instances defined by the resources defined in the domain model.

In order to efficiently generate all of these different types of targets it is important that SDK be designed in a modular fashion.  SDK generators should be built in such a manner as to facilitate the easy inclusion of new generators to suit the purpose at hand.  This should be configurable.  
  
Illustrated below is an example of one such high level modular code generation architecture:

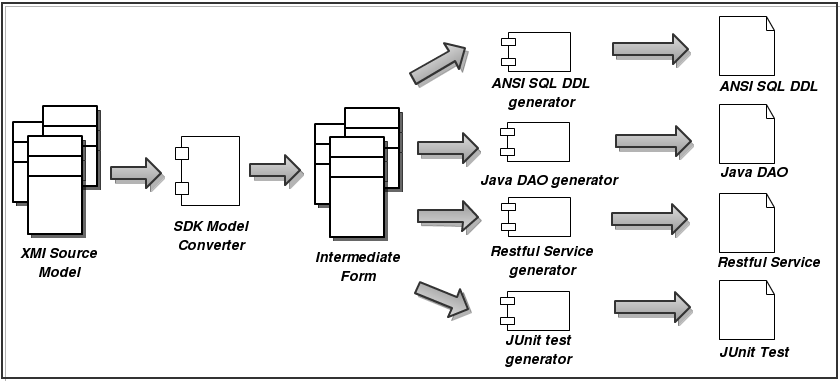


Figure : High level candidate architecture for the new SDK.

In this architecture, a domain model represented in XMI is translated by an XMI specific SDK model converter into an intermediate form.  All generators are familiar with this intermediate form construct and can create their own specific target artifacts independently of the other generators.  An added benefit of this approach is that generators can use other generators to generate fragments of code, which can then be combined to create a composite generated code fragment.  For instance, a DAO code generator may rely on an ANSI SQL code generator to generate the ANSI SQL that it will use in its DAO methods. Of course this reuse need not be limited to two generators. It would be completely acceptable for whole chains of generators to use each other in this fashion.

Another advantage of using this pluggable generator approach is that new platforms can be supported without changing support for existing platforms. As an example, the same intermediate form that was used to generate a web based application could also be used to generate an application for Google’s Android mobile phone platform. Furthermore, that Android application generator could be created without any support from the SDK development team. Along the same lines subsequent versions of the generated Andriod platform could be produced by generators that are Android platform version aware. This represents a significant step forward over the existing SDK code generating framework.  
  
To support this proposed chain generating capability on given domain in a domain model, all artifact generators will have to employ a regular generation interface.  That interface is defined by the following function:

**If there exists a “D” where “D” is a domain model, and there also exists a “d” where “d” is a domain object that is a member of the domain model “D”.  Let “f” be a code generating function.**

**Then,**

***f (d,D) = s***

**where “s” is a string representing the resulting generated artifact based on “f”, “d” and “D”.**

Every SDK artifact generator should implement this function.

## Multiple Language Support

Some consideration should be paid to the generation of multiple language sets.  Currently, Java is the only target language supported by the SDK.   There is really no technical reason PHP, Ruby, JavaScript, Python, C++, or other languages are not supported by the SDK; the approach of creating an intermediate form completely supports the generation of multiple language targets.  Consequently, if it is found to be beneficial to support multiple language platforms, in order to support computable operability amongst those generated platforms, the accompanying generated domain representations and domain transport protocols will need to be in text. This certainly calls into question some of the approaches SDK is using today; Spring Remoting is only feasible as a communications device amongst applications executing within the Java environment.  Java should be the initial target environment, but SDK should align its design direction with one that could easily support multiple language platforms.

## Presentation

Presentation, like security, is a broad ranging topic.  SDK will support out of the box (X)HTML based presentations.  These will include RDFa, microformats, and Atom. Within this technology landscape, SDK will utilize CSS along with dynamic HTML to provide a rich application experience for human consumers.  These same constructs will still be consumable by computing processes as they will contain semantically rich information via RDFa or microformats.  The goal of SDK is to make sure that all of its generated presentations serve both humans and computers.  This will eliminate the need for the service caller to switch calls to accommodate end user specific representations.  
  
SDK’s modular generator architecture will support the addition of new generators by developers to meet their particular presentation requirements, whether it is for a .NET desktop application or a mobile phone client.

## Concepts

SDK will be concept aware.  Developers will be able to annotate the SDK intermediate form model objects with concept information, and this information will be made available in generated artifacts whenever possible.  For instance, all (X)HTML representations created by SDK generated services will convey concept information from developer chosen terminology services.  The RESTful services will portray this concept information whenever the HTTP method OPTIONS is invoked.  SOAP based services will represent concept information using WSDL-S. This feature will support both automated and manual discovery of services.

## Documentation and Artifacts

SDK will generate documentation for all the services it creates.  This documentation will serve automated and manual service API discoverability, enterprise governance considerations, and technical support.  Whenever the model changes, upon regeneration, these changes will be automatically populated in the generated documentation.  The following is a non exhaustive list of the types of documentation that will be automatically generated by SDK.

* Language API documentation
* Service API documentation
* Class diagrams
* Conceptual model text
* Conceptual model diagrams
* Logical model text
* Logical model diagrams
* Physical model text
* Physical model diagrams
* Security roles and authorization

Due to SDK’s extensible generator infrastructure, developers could employ other documentation generators to meet their particular governance and or policy requirements.

## Validation

The SDK intermediate form will support the notion of object and data constraints.  From these model entities validation logic will be generated in the target language of choice.  This will cover both syntactic (format, length, range, type, and presence) and semantic (inter property dependencies, relationship constraints). Developers will be able to reuse, extend, or replace these generated validation artifacts as they see fit.

## Performance

Performance is of paramount concern.  There will be three design goals met by the SDK design team.

1. **SDK will generate fast executable code -** All code generated by SDK will come paired with automated instrumentation code.  This instrumentation code will allow developers to test the performance of critical regions of SDK generated services. These critical regions will be focused around data store persistence and retrieval, serialization of objects, presentation of information, and network traffic.
2. **SDK will generate extensible code** - Service artifacts generated by SDK will be created so as to allow developers easy means for overriding the code used by a particular service, without changing the overall architecture. Developers will then be able to address performance issues that are unique to their particular domain model or application.
3. **SDK will generate services that follow RESTful considerations** - This will allow system operation teams to apply content delivery techniques such as using a CDN or cache proxy to improve deployment performance.  This can be applied without the need for developer involvement and as such can represent a more financially palatable solution for improving service performance. Note that the creation of RESTful services does not mean that SOAP will not be supported.

# Services

A service is a handle that grants access to a computable operation on a set of resources.  Services may be implemented as methods on an object, a defined procedure on a data store item, or a URI that is backed by a server side executable.  A caller of a service is a client that has a handle to the service and has passed parameters to the service via that handle.  Callers of services may or may not expect to receive information in return.  Services may be enacted across networks; service callers do not have to reside within the same memory space as the service host.  
  
Services are used by businesses and organizations to provide business functionality to their customers.  The most popular form of service architectures are web based.  SOAP was considered the web service approach of choice for most enterprise deployments; more recently there has been industry wide movement to URI based services.  In particular, Representational State Transfer (REST) architectures have become popular.  Attempts to move in this direction have resulted in a menagerie of home grown services that may be naively described as RESTful; REST as an architectural style has several constraints that must be adhered to in order for a web service to be considered a RESTful web service.

The following sections will address how SOAP and REST can work in the enterprise, and makes the case that SDK can generate both types of services. REST is an architectural style, and SOAP is an application protocol, so direct comparisons are not reasonable. Hypertext transport Protocol (HTTP) is a restful protocol implementation of web services; an HTTP based approach to creation of web services is provided as an example.

## SOAP based Web Services

SOAP is a “structured information passing” specification that found its origins in XML-RPC as devised by Dave Winer and others from Microsoft. Its principal purpose is to facilitate message passing via Internet application layer protocols like SMTP and HTTP, the latter of which is the most popular. It is due to this popularity that SOAP via HTTP is also referred to as just SOAP.

The SOAP specification[[10]](#footnote-10) consists of four parts:

* Processing Module – describes how a SOAP message is to be processed. This includes defining the callers and listeners in SOAP networks, as well as defining rules surrounding the forwarding of messages from SOAP senders to SOAP receivers.
* Extensibility Module – describes SOAP features and SOAP modules.
* Protocol Binding – describes the binding to an underlying Internet application protocol.
* Message Construct – describes how SOAP messages are constructed.

The underlying principal behind SOAP is to expose a server side object’s attributes and methods to a calling party. SOAP pushed the functional aspect of language programming to a network environment. SOAP binding specifications support the definition of objects, operations on those objects, and variable binding on those operations.

## Web Service Discovery via WSDL-S

The W3C has also created a specification for describing web services. The Web Service Description Language (WSDL) is an XML based representation to be used to define document or procedure based services on the web. It is closely associated with SOAP. WSDL does not contain any built in support for represent concepts, however, a specification called WSDL- S (“S” stands for semantic) out of the University of Georgia provides WSDL based support for annotating WSDL with concepts and terminologies. This specification was created in 2005, and was designed to allow developers to annotate their web services using the ontology or semantic representation language of their own choice. The SDK team will support the generation of WSDL-S service descriptions whenever appropriate.

## RESTful Web Services via HTTP

REST is an architectural style that was formally described in a doctoral dissertation written by Roy Thomas Fielding of the University of California, Irvine in the year 2000[[11]](#footnote-11).  Fielding was one of the principal authors of the HTTP specification (RFC 2616). In his dissertation, Fielding delineated several properties of scalable distributed systems, and described them as representational state transfer, from which the acronym REST was created.  
  
In the rest architecture, clients and servers may transfer the state of resources back and forth.  Resources could be described as a concept, and an instance of a resource is identified via a URI.  Resources can be related to other resources via links. Links are constructs that describe pointers from resource to resource.  The state of a resource is captured by its representation.  Servers also use representations to tell the client of its allowed state transitions from a particular resource.  State transitions can be defined as the application of a particular operation on a resource.  By reading the resource’s representations, clients are able to navigate a group of related resources within an application.  It is through this navigation of resources via their representations that a client can get work done on the server.  By design, this approach to designing services promotes the use of a small and regular set of operations, over a large set of defined resources.  The list of RESTful design considerations are summarized below.

1. **Resources identified by URIs** - All resources in the application are addressable via URIs.
2. **Representations of resources** - All resources have a representation that accurately conveys the current state of the resource.  Resources may have multiple representations.
3. **Small, but regular group of allowed operations** - For every given resource there are defined the same small set of resource operations.
4. **Hypermedia as the engine of application state (HATEOAS)** - Links embedded in representations indicate the allowed state transfers for a given resource.

This description of a client to server exchange fits neatly into the world of domain models.  Domain objects map directly to resources.  Relationships represent links.  State transitions fit into operations on resources.  URIs are unique identifiers of resources.  Object instances are defined by representations.  At the highest levels, the REST architectural approach for creating services aligns well with the creation of applications via model driven engineering.

## RESTful Service Discovery via HTTP

The HTTP application protocol can be used to provide RESTful services[[12]](#footnote-12).  HTTP relies on URIs to identify resources.  Text based representations like RDFa can represent resources.  The number of operations allowed via HTTP is a small and finite set; GET, HEAD, PUT, POST, DELETE, TRACE, CONNECT, and OPTIONS.  For normal persistent store operations, create, read, update, and delete (or CRUD), the HTTP methods POST, GET, PUT, DELETE are sufficient.  To support service discovery, the HTTP method OPTIONS is used.  According to the HTTP protocol, for a given caller, invoking the OPTIONS method on a resource’s URI shall return a list of allowed methods for that resource.  The following mock invocation example illustrates this concept:

**# Request**

**OPTIONS /patient HTTP/1.1**

**Host:** [**http://www.mayoclinic.com**](http://www.mayoclinic.com)

**# Response**

**HTTP/1.1 204 No Content**

**Allow: OPTIONS, POST, GET**

**Link: <http://www.mayoclinic.com/documents/patient>; type=text/html; rel=help**

In this example the URI that represents the list of patients at the Mayo Clinic is ***http://www.mayoclinic.com/patient***.  For this user, the HTTP methods GET, POST, and OPTIONS are allowed.  This is indicated by the standard HTTP header “Allow”. The method GET being allowed means that this user is allowed to read the list of patients at the clinic.  The POST method means the user may add a new patient to the list of patients.  The absence of the DELETE method means that this user is not allowed to delete patient information.  All of this information is conveyed over the network as HTTP headers, as indicated by the standard HTTP response code 204.  
  
Note that for completeness, the OPTIONS method is returned as an allowed method.  If the OPTIONS method was not allowed, the server could simply return a response with the HTTP response code 403 Forbidden, or simply return no information at all.  Also of importance is the information returned in the “Link” header.  This header points to an additional link that gives more information about this /patient URI.  This is where documentation reflecting the representation format of a patient resource could be provided. The “rel” attribute of the link indicates that this URI points to developer help for this service.  Note that a URI to the concept “Patient” as defined by a terminology service could be provided here as well.  
  
HTTP provides a built in mechanism for promoting automated service discovery.  Services based on HTTP, linked to a terminology service, and using link rich RDFa representations to illustrate resources could support automated service discovery, thereby creating a means for both manual and automated semantic interoperability.

## SOAP over REST

With REST forming the basis of the preferred approach to service generation for SDK, what does this mean for SOAP based web services?  At its surface, SOAP represents in many ways a reimplementation of many of the features already provided by HTTP, but in a non-restful manner.  SDK can continue to support SOAP based services by simply implementing those RESTful CRUD operations as SOAP calls.   In other words, SOAP calls will act as a proxy for the underlying RESTful architecture.  Of course, WSDL-S will still be generated and deployed for the benefit of both RESTful and SOAP clients.

# SDK Development Approach

Building the next generation SDK will certainly be a challenge. Luckily, there are many open source tools for accomplishing many of the goals outlined in this document.

## Reuse existing infrastructure

One way to approach this is to cobble together a best of breed of persistence mechanisms, service frameworks, and presentation tools.  This approach will lead to some early gains.  Indeed, this is the approach followed by the current version of SDK.  This approach has some drawbacks.  One of the drawbacks is that the SDK team is challenged to manage all the different versions of software that are tied together to meet its requirements.  It is not a guarantee that the current combination of libraries will continue to work together as new releases of software occur.  Developers wanting to move forward to adopt new functionality for their own applications are sometimes hampered by SDK’s own release schedule.  Maintaining backward compatibility while juggling all of these different libraries has proven to be difficult.

## Build Our Own by Copying Others

An alternative approach would be to build a standalone version of SDK that does not rely on popular frameworks or libraries.  This standalone SDK library will have its own upgrade cycle separate and apart from the popular development tools used by other developers.  This will require more effort from the development team; this work could be mitigated by simply extracting, refactoring, and repackaging the most relevant parts of popular open source tooling frameworks[[13]](#footnote-13).  As their software licenses provide, the SDK team will repackage and re-factor open source code to fit into a standalone SDK library.  This approach will ultimately result in an SDK software package that is easier to use, maintain, and upgrade.  It is this improvement in developer friendliness that will help drive adoption of the SDK.  This would allow SDK developers to spend more time providing the new functionality needed to support semantic interoperability, and less time addressing arcane library upgrade issues.

## Candidate SDK Intermediate Form - EMF Ecore

For practical purposes, the SDK team will chose EMF Ecore as the candidate for the new SDK artifact generator’s intermediate form.  As mentioned in an earlier treatment, Ecore’s model satisfies all the requirements for a meta model.  It can represent classes, properties, operations, and bi-directional relationships as first class model constructs.  Ecore’s lack of first class support for concepts is a small issue that could be addressed by the use of annotations.  In Ecore, annotations can be applied to classes, relationships, properties, and operations.   In addition, the other benefits of using Ecore cannot be overlooked, in that Ecore provides a code generation facility that can be used as a template for the creation of the SDK generator.  As mentioned before Ecore has both Java community and industry backing.  The Ecore model can be represented in other meta modeling formats such as OWL.  This would allow us to use this model outside of the Java world should there be a need to do so.

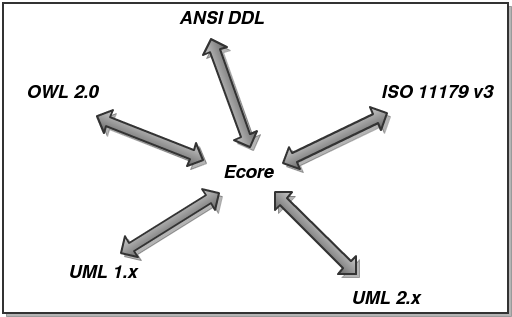


Figure : Ecore as a universal intermediate form for meta model translation.

## Challenges

Although Ecore is a full fledge code generation utility, attaining reuse by extending it can prove to be a challenge.  Outside of the usual upgrade path considerations, the Ecore generator has several limitations that make it a non-ideal candidate for total reuse for the new SDK platform.  The issues are as follows:

* Ecore generates only Java related artifacts.  It does not readily allow for the generation of artifacts in other languages.  This does not allow Ecore to accommodate all developers, since generation of multiple language artifacts is not supported.  An instance where this would be an issue would be in the generation of JavaScript for browser based form validation.
* Ecore uses the JET template system to generate code.  JET is not a pure presentation template system[[14]](#footnote-14), and is very Java centric in nature.  Moreover, there are more suitable open source templating systems available that are also multi platorm in nature.  One such example is StringTemplate[[15]](#footnote-15), which runs on both Java and Python.
* Ecore’s default Java generation artifacts are not designed to facilitate use outside of the Ecore environment.  SDK will create artifacts that can be used outside of the code generation and model frameworks used to create the code.  This source level dependency that cannot be broken without refactoring Ecore itself.  Although we will be using SDK to enhance semantic interoperability, we do not wish the SDK itself to inadvertently become a hamper to the sharing of its generated language specific constructs.

Instead of reusing Ecore as it is, SDK should use Ecore as a template for designing a new Eclipse based code generation utility that completely meets the goals of the SDK developer community.  The only direct dependency would be supporting the Ecore model representation.  This will allow SDK to take advantage of the model to model translation facilities present in Ecore, while still improving on the existing code generation abilities of Ecore itself.

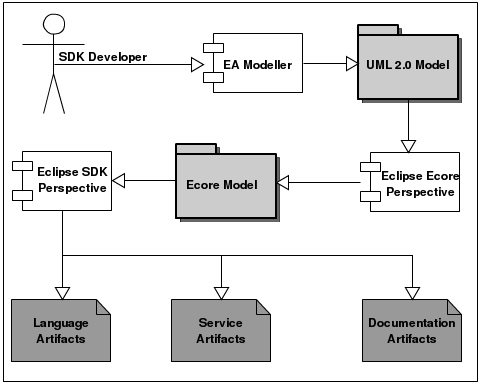


Figure : Sample development workflow involving model translation and artifact generation.

## Solution

Instead of reusing Ecore as it is, SDK should use Ecore as a template for designing an Eclipse based code generation utility that completely meets the goals of the developer community.  The only direct dependency would be supporting the Ecore model representation.  This will allow SDK to take advantage of the model to model translation facilities present in Ecore.

# Enterprise Compliance

The Enterprise Conformance and Compliance Framework (ECCF) is a framework for documenting and confirming compliance for services created within the scope of the Services Aware Inter-operability Framework (SAIF) guidelines.  ECCF specifies several documents that must be created in order for a developed application to be deemed in compliance.  The documents that are to be created fall within three levels; Conceptual Information Model (CIM), Platform Independent Model (PIM), and the Platform Specific Model (PSM).  At the ECCF website there are several examples of documentation that fall within these definitions.  Upon careful analysis of the content expected to be made available in these documents, it is apparent that much of the information required could be generated by SDK.  Indeed, given the proposed high level architecture of SDK, the creation of ECCF artifacts could be accomplished by simply creating an ECCF generator.  It is the expectation of the ECCF team that automated generation would supplement, and in some cases replace, the manual creation of these documents.  
  
The following is a list of the sample documents that the SDK team has identified as candidates for some automated generation:

* [ConformanceStatements.doc](http://gforge.nci.nih.gov/svnroot/candc/trunk/documents/artifact_templates/conceptual/ConformanceStatements.doc)
* [caBIG\_Service\_Integration\_Guide\_Template.doc](http://gforge.nci.nih.gov/svnroot/candc/trunk/documents/artifact_templates/integration/caBIG_Service_Integration_Guide_Template.doc)
* [caBIG\_Platform\_Independent\_Model\_Template.doc](http://gforge.nci.nih.gov/svnroot/candc/trunk/documents/artifact_templates/logical/caBIG_Platform_Independent_Model_Template.doc)
* [caBIG\_Platform\_Specific\_Model\_Template.doc](http://gforge.nci.nih.gov/svnroot/candc/trunk/documents/artifact_templates/platform/caBIG_Platform_Specific_Model_Template.doc)
* [caBIG\_Computational\_Independent\_Model\_Template.doc](http://gforge.nci.nih.gov/svnroot/candc/trunk/documents/artifact_templates/conceptual/caBIG_Computational_Independent_Model_Template.doc)
* [caBIG\_Conceptual\_Functional\_Application\_Specification\_Template\_010.doc](http://gforge.nci.nih.gov/svnroot/candc/trunk/documents/artifact_templates/conceptual/caBIG_Conceptual_Functional_Application_Specification_Template_010.doc)

The advantage of this approach cannot be overlooked.  Keeping documentation in sync with source code and deployed systems has long been the bane of software development projects.  Whenever possible, the application of smart development methods like code generation can eliminate duplication of effort and labor intensive artifact creation.  Using SDK to fulfill these tasks can result in tremendous savings for the organization on the whole.

Another goal of SAIF is to automatically verify compliance of a service at runtime. SDK will support this consideration; automated generation of semantically aware smoke tests can verify the correctness of a running SDK generated service.

# Open Health Tools

Open Health Tools (OHT) is “an open source community with a vision of enabling a ubiquitous ecosystem where members of the Health and IT professions can collaborate to build interoperable systems that enable patients and their care providers to have access to vital and reliable medical information at the time and place it is needed”. Their vision is the creation of a system that facilitates international health exchange for improving patient services. There operating model is based on that of the Eclipse Foundation, in that they are committed to enroll the open source development and corporate communities to enable the creation of a freely available toolset that will enable the creation of semantically aware and interoperable health care applications.

OHT membership is free; prospective members can apply for board participation by filling out a form. Membership is subject to scrutiny, and the ability of a prospective member to contribute meaningfully by way of financial gifts and/or technical contribution is weighed heavily.

OHT’s organizational goals align with the goal of the SDK team to create a semantically aware artifact generator. SDK could align itself strategically with OHT by offering its artifact generator as a way to create semantically aware services. The OHT is also working on the creation of a reference architecture that supports semantic interoperability. The SDK code provides a working generator set that will generate applications that are in compliance with the OHT reference architecture.

# Conclusion

SDK represents a monumental achievement in the application of code generation to improve the development quality of our developer base. Much has been learned by this effort. Given the new direction set by this organization with regards to semantic interoperability, SDK remains well poised to contribute positively by providing developers with more tools that support working operability. To do this SDK must embrace a new technological approach. Such an approach will take advantage of both established and emerging industry standards. A big part of this effort will be utilizing the technologies available in the Eclipse framework. Leveraging EMF technology will provide a boost to the development of a more advanced, yet more user friendly SDK. By favoring convention over configuration, the new SDK will prove to be big ally to this organization’s development community.

# References

No authors credited. “Eclipse Modeling EMF – Home” The Eclipse Foundation

July 9th, 2010 http://www.eclipse.org/modeling/emf/?project=emf

Dave Raggett, Arnaud Le Hors, and Ian Jacobs. “Links.” W3C.

July 9th, 2010 <http://www.w3.org/TR/html401/struct/links.html>

Subbu Allamaraju. “RESTful Web Services Cookbook.” O’Reilly Media, Inc. 2010.

Roy Thomas Fielding. “Architectural Styles and the Design of Network-based Software Architectures.” University of California, Irvine Campus

July 7th, 2010. <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

Many authors. “Document.” Wikipedia.

July 13th, 2010. <http://en.wikipedia.org/wiki/Document>.

[Dolin RH](http://www.ncbi.nlm.nih.gov/pubmed?term=%22Dolin%20RH%22%5BAuthor%5D), [Alschuler L](http://www.ncbi.nlm.nih.gov/pubmed?term=%22Alschuler%20L%22%5BAuthor%5D), [Beebe C](http://www.ncbi.nlm.nih.gov/pubmed?term=%22Beebe%20C%22%5BAuthor%5D), [Biron PV](http://www.ncbi.nlm.nih.gov/pubmed?term=%22Biron%20PV%22%5BAuthor%5D), [Boyer SL](http://www.ncbi.nlm.nih.gov/pubmed?term=%22Boyer%20SL%22%5BAuthor%5D), [Essin D](http://www.ncbi.nlm.nih.gov/pubmed?term=%22Essin%20D%22%5BAuthor%5D), [Kimber E](http://www.ncbi.nlm.nih.gov/pubmed?term=%22Kimber%20E%22%5BAuthor%5D), [Lincoln T](http://www.ncbi.nlm.nih.gov/pubmed?term=%22Lincoln%20T%22%5BAuthor%5D), [Mattison JE](http://www.ncbi.nlm.nih.gov/pubmed?term=%22Mattison%20JE%22%5BAuthor%5D). “HL7 clinical Document Architecture.”

July 12th, 2010. http://www.ncbi.nlm.nih.gov/pubmed/11687563

No author credited. “Link Relations“. Internet Assign Numbers Authority (IANA).

July 9th, 2010. <http://www.iana.org/assignments/link-relations/link-relations.xhtml>

Rama Akkiraju, IBM Research et al. “Web Service Semantics - WSDL-S”

July 15th, 2010. <http://www.w3.org/Submission/WSDL-S/>

[Erik Christensen](mailto:erikc@microsoft.com), [Francisco Curbera](mailto:curbera@us.ibm.com), [Greg Meredith](mailto:gregmer@microsoft.com), [Sanjiva Weerawarana](mailto:sanjiva@us.ibm.com). “Web Services Description Language (WSDL) 1.1”

July 15th, 2010. <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>

Multiple authors. “OpenID Authentication 2.0 – Final”

July 15th, 2010. <http://openid.net/specs/openid-authentication-2_0.html>

No author credited. “Open Health Tools Tooling Architecture Vision”, Open Health Tools

July 21st, 2010. <http://www.openhealthtools.org/Reports/Apr08/OHTToolingArchitectureVisionV2.pdf>

Kerstin Altmanninger, Petra Brosch, Gerti Kappel, Philip Langer, Martina Seidl, Konrad Wieland, and Manuel Wimmer. “Why Model Versioning Research is Needed!? An Experience Report\*”

July 22nd, 2010. <http://publik.tuwien.ac.at/files/PubDat_177761.pdf>

# Appendix

## SDK v. SDK Feature Matrix

There are several features that are available in the new SDK but will not be available in the old SDK. The following table illustrates how the new SDK compares with the old SDK.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **New SDK** | **Old SDK** | **Comments** |
| Open intermediate form | Yes | No | New SDK new intermediate form will be based on Ecore. Ecore models can be persisted as XMI documents. |
| Support for multiple language. | Yes | No | New SDK’s pluggable generator design will allow developers to create generators that can create artifacts in any programming language. The current SDK supports Java only. |
| Support for non code based artifacts. | Yes | Partial | New SDK will allow for generation of artifacts that are not language specific. Developers can create pluggable generators that can be used to create diagrams, documents, or other non language specific artifacts. Current SDK does not support this without required build changes. |
| True REST service generation. | Yes | No | New SDK will create true RESTful implementation by supporting RESTful object representations. Current SDK does not fully meet all the REST constraint requirements. |
| Open transport protocol. | Yes | No | New SDK will generate a reference implementation that is uses HTTP application protocol to transport information, thus enabling cross platform support. Current SDK relies on Spring Remoting, and as such can only be used within the Java platform. |
| Pluggable generator architecture. | Yes | No | Users of the new SDK can write their own pluggable generators. |
| Pluggable model translation architecture. | Yes | No | Users of the new SDK can write their own pluggable model translators. |
| GUI based IDE interface. | Yes | No | New SDK will rely on the Eclipse IDE to provide GUI access to its main functions. |
| Service based interface. | Yes | No | New SDK will provide a object based service interface to expose SDK functions to new applications. |
| Command line interface. | Yes | Yes | Both the current and new SDKs will support command line invocation of functionality. |
| Generation of automated testing compliance. | Yes | No | New SDK will generate unit, regression, and smoke tests for its generated reference implementation. |
| Full concept integration. | Yes | Partial | New SDK will support concepts as a modeling artifact. Users of SDK will be able to assign concepts to classes, relationships, properties, and operations. |
| Read write Rational Rose model | Yes | No | New SDK will rely on EMF tooling support for reading and writing Rational Rose model files. |
| Read write Enterprise Architect model | Yes | Partial | New SDK will rely on EMF tooling support for reading EA model files. SDK developers will create support for writing XMI files that can be read by EA. |
| Read write database specific DDL | Partial | No | SDK will support partial reading and writing of DDL scripts for, but not limited to, Oracle, SQLServer, Sybase, PostgreSQL, and MySQL. |
| Support for multiple language implementations | Yes | No | Users of new SDK can create generators in programming languages supported by JSR-223 language interpreters. |

1. With the future in mind, the change from “Software Development Kit” to “Service Development Kit” was deliberate. [↑](#footnote-ref-1)
2. The full treatment of links as defined by W3C is found here http://www.w3.org/TR/html401/struct/links.html [↑](#footnote-ref-2)
3. <link href=”http://www.social.org/person/malik” rel=”brother” rev=”sister”> My Brother <link/>. [↑](#footnote-ref-3)
4. More information about the Link Relation Registry can be found here: http://www.iana.org/assignments/

   link-relations/link-relations.xhtml [↑](#footnote-ref-4)
5. The current version of SDK supports a subset of ISO 21090. This support will be continued in the future

   release. Additional datatypes may be supported if dictated by the organizational directive. [↑](#footnote-ref-5)
6. Operations are not relevant to this application as the conveyance of information regarding the structure of the domain object is the only consideration. [↑](#footnote-ref-6)
7. Full definition of a document can be found here: http://en.wikipedia.org/wiki/Document. [↑](#footnote-ref-7)
8. HL7 clinical document architecture can be found here http://www.ncbi.nlm.nih.gov/pubmed/11687563 [↑](#footnote-ref-8)
9. URIs are being replaced with the more internationally friend International Resource Indicator (IRI). [↑](#footnote-ref-9)
10. More information on the SOAP specification can be found here: http://www.w3.org/TR/soap/ [↑](#footnote-ref-10)
11. See Architectural Styles and the Design of Network-based Software Architectures at http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm [↑](#footnote-ref-11)
12. In Subbu Allamaraju’s “RESTful Web Services Cookbook” he provides an excellent treatment on HTTP

    methods. In particular the use of the OPTIONS method for service discovery is addressed in chapter

    “14.2 How to Use OPTIONS”. [↑](#footnote-ref-12)
13. For instance for persistence SDK could extract, refactor, and reuse a fully adapted SDK version of Spring’s JDBCTemplate persistence package. [↑](#footnote-ref-13)
14. The JET template system supports the use of programming logic in templates. This breaks the model view controller separation paradigm and as such could encourage the creation of coupled template presentation and creation logic. [↑](#footnote-ref-14)
15. Terrence Parr is the creator of StringTemplate. He is a avid compiler theorist and implementer,

    and StringTemplate benefits much from his research. More about StringTemplate can be found at

    www.stringtemplate.org. [↑](#footnote-ref-15)