Transatlantic TUmour MOdel Repositories

D4.1.1
State of the art review on web-services, interoperability design directions tailored to TUMOR needs

Project Number:	FP7--IST-247754
Deliverable id:	D4.1.1
Deliverable name:	State of the art review on web-services, interoperability design directions tailored to TUMOR needs
Submission Date:	31/3/2011

Information Society Technologies

COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	TUMOR
Project Full Name:	Transatlantic TUmour MOdel Repositories
Document id:	D4.1.1
Document name:	State of the art review on web-services, interoperability design directions tailored to TUMOR needs
Document type (PU, INT, RE)	PU
Version:	7
Submission date:	
Editor: Organisation: Email:	Thomas Deisboeck MGH DEISBOEC@HELIX.MGH.HARVARD.EDU

Document type PU = public, INT = internal, RE = restricted

ABSTRACT: This deliverable describes the implementation of dedicated web services (using WSDL and SOAP) that provide various facilities for connecting data and tools between the EU repository and the MGH-CViT. Services will be defined both for the CViT DMR and the EU repository, and accessible through the TUMOR integrated environment. These will include functionalities such as the ability to locate and search the stored models, review their attributes, download the models, upload new models into the repository, and execute the models. Much of this is already available or under development for CViT (e.g. https://www.cvit.org/node/314 [Data service]); more work will be required to implement equivalent services for the EU repository, and this will be done in close collaboration with T3.1. These services will be designed with consistent APIs to facilitate their combination within TUMOR, such that information from both repositories can be combined. For example, the system will allow users to perform a simulation of a coupled model with a component from each repository, parameterised using data from each repository. Development of these services will make use of existing tools where appropriate, for example, CViT provides its existing RDF repository and semantic modelling and annotation facilities, the caGrid domain information model and Grid service API, and has a model execution framework currently under development.

KEYWORD LIST: design directions, state-of-the-art review, web services

MODIFICATION CONTROL			
Version	Date	Status	Author
1	March 1, 2011	Draft	Thomas Taylor
2	March 7, 2011	Draft	Thomas Deisboeck, Thomas Taylor
3	March 14, 2011	Draft	Thomas Deisboeck, Thomas Taylor
4	March 18, 2011	Draft	Thomas Taylor
5	March 24, 2011	Draft	Thomas Taylor
5.1	March 29, 2011	Draft	Stelios Sfakianakis
6	March 30, 2011	Pre-final	Thomas Taylor
7	March 31, 2011	Final	Vangelis Sakkalis, Thomas Taylor

List of Contributors

– All consortium
– Thomas Deisboeck (MGH)
– Thomas Taylor (MGH-INFOTECH Soft)
– Stelios Sfakianakis (FORTH)
– Vangelis Sakkalis (FORTH)
– Giorgos Zacharioudakis (FORTH)

Contents

1 EXECUTIVE SUMMARY 5
1 TASK DESCRIPTION 6
2 WEB SERVICES DISCUSSION 7
COMPARISON SOAP VS. REST 7
CVIT DIGITAL MODEL REPOSITORY APPROACH 8
BRIEF SURVEY OF SELECTED WEB SERVICES ON THE INTERNET 9
3 REVIEW OF WEB SERVICES TECHNOLOGIES 13
WEB SERVICE STYLES OF USE 14
WEB SERVICES DESCRIPTION LANGUAGE (WSDL) 15
3.1.1 Service Description 15
3.1.2 Component Model 15
3.1.3 Types 20
SIMPLE OBJECT ACCESS PROTOCOL / SERVICE ORIENTED ARCHITECTURE PROTOCOL (SOAP) 21
3.1.4 SOAP Messages 21
3.1.5 SOAP - Remote Procedure Calls 22
SEMANTIC ANNOTATIONS FOR WSDL AND XML SCHEMA (SAWSDL) 23
3.1.6 Annotation Mechanisms 23
3.1.7 Annotating WSDL 23
3.1.8 Annotating XML Schema 24
REPRESENTATIONAL STATE TRANSFER (REST) WEB SERVICES 28
3.1.9 Data Elements 29
3.1.10 Resources and Resource Identifiers 29
3.1.11 Representations 29
3.1.12 Connectors 30
3.1.13 Components 30
WEB APPLICATION DESCRIPTION LANGUAGE (WADL) 32
4 MGH CENTER FOR THE DEVELOPMENT OF A VIRTUAL TUMOR (CVIT) OVERVIEW 34
MGH-CVIT DMR AND CMEF COMBINED ARCHITECTURE 34
MGH-CVIT DMR AND CMEF (COMBINED DOMAIN MODEL) 36
MGH-CVIT DMR TUMOR WEB SERVICE 54

1 Executive Summary

The TUMOR project aims at developing a European clinically oriented semantic-layered cancer digital model repository from existing EC projects that will be interoperable with the US grid-enabled semantic-layered digital model repository platform at CViT.org (Center for the Development of a Virtual Tumor, Massachusetts General Hospital (MGH), Boston, USA) which is NIH/NCI-caBIG compatible. This interoperable, CViT interfaced, environment will offer a range of services to international cancer modellers, bio-researchers and eventually clinicians aimed at supporting both basic cancer quantitative research and individualized optimization of cancer treatment. This 'Transatlantic' project will therefore be the starting point for an international validation environment that will support joint applications, verification and validation of the clinical relevance of cancer models.

The purpose of this deliverable is to review the state-of-the-art web service technologies and service description languages capable of connecting data and tools between the EU repository and the MGH-CViT. Services will be defined both for the CViT DMR and the EU repository, and accessible through the TUMOR integrated environment. These will include functionalities such as the ability to locate and search the stored models, review their attributes, download the models, upload new models into the repository, and execute the models. Much of this is already available or under development for CViT (e.g. https://www.cvit.org/node/314 [Data service]); more work will be required to implement equivalent services for the EU repository, and this will be done in close collaboration with T3.1

There are also open source CellML, SBML and FieldML simulation tools either already available or under development within VPH projects, with efforts co-ordinated by the VPH NoE, which will be utilised by the model execution server. Key to achieving interoperability is the use of semantic meta-information associated with both models and data, as indicated in the description of T3.1. Annotation of both models and data with terms from the same ontology (or mapped ontologies) will allow for mechanistic determination of corresponding entities (for example, variables in two models referring to the same biological entity, or models describing the same process). In many cases these annotations will refer to third party web-service enabled databases, such as MINT, IntAct, DIP,BioCyc, KEGG, PUMA2, or Reactome, and in such cases the use of these databases is likely to be useful in integrating models. Thus the use of semantic metadata together with the web services developed will facilitate registration, discovery, comparison and interoperability between models, and the integration of data from clinical, laboratory and modeling domains.

1 Task Description

T4.1 Interoperable interface and connectivity design (PM8-PM28) (Leader: 5, Participants: all)

This task will implement dedicated web services (using WSDL and SOAP) providing various facilities for connecting data and tools between the EU repository and the MGH-CViT. Services will be defined both for the CViT DMR and the EU repository, and accessible through the TUMOR integrated environment. These will include functionalities such as the ability to locate and search the stored models, review their attributes, download the models, upload new models into the repository, and execute the models. Much of this is already available or under development for CViT (e.g. https://www.cvit.org/node/314 [Data service]); more work will be required to implement equivalent services for the EU repository, and this will be done in close collaboration with T3.1. These services will be designed with consistent APIs to facilitate their combination within TUMOR, such that information from both repositories can be combined. For example, the system will allow users to perform a simulation of a coupled model with a component from each repository, parameterised using data from each repository. Development of these services will make use of existing tools where appropriate, for example, CViT provides its existing RDF repository and semantic modelling and annotation facilities, the caGrid domain information model and Grid service API, and has a model execution framework currently under development.

2 Web Services Discussion

We need to decide (among other things) the technologies and the standards that will be used for the implementation of the interoperable interfaces. As mentioned in the Description of the Work these interfaces will be "service oriented" and particularly based on Web Services. As mentioned in Wikipedia, W3C defines two major types of web services:

- REST type web services, provide a uniform interface and they are more "web-friendly". For example, an HTTP(S) REST web service will use some or all of the standard HTTP protocol operations: GET, POST, HEAD, PUT, DELETE,...
- SOAP/WSDL based Web Services (also known as "Big web services") that provide an arbitrary (i.e. developer defined) set of operations

There are pros and cons to both of them. So we need either to choose one of the two or maybe follow a hybrid approach. In any case these decisions should be well documented and justified.

--Stelios 19:55, 16 February 2011 (EET)

Comparison SOAP vs. REST

http://www.ajaxonomy.com/2008/xml/web-services-part-1-soap-vs-rest

http://www.petefreitag.com/item/431.cfm

SOAP	**REST-XML**
Pros: • Language, platform, and transport agnostic • Designed to handle distributed computing environments • Is the prevailing standard for web services, and hence has better support from other standards (WSDL, WS-*) and tooling from vendors • Built-in error handling (faults) • Extensibility • Support for semantic web service Cons: • Conceptually more difficult, more "heavy-weight" than REST • More verbose • Harder to develop, requires tools	Pros: • Language and platform agnostic • Much simpler to develop than SOAP • Small learning curve, less reliance on tools • Concise, no need for additional messaging layer • Closer in design and philosophy to the Web Cons: • Assumes a point-to-point communication model--not usable for distributed computing environment where message may go through one or more intermediaries • Lack of standards support for security, policy, reliable messaging, etc., so services that have more sophisticated requirements are harder to develop ("roll your own") • Tied to the HTTP transport model

CViT Digital Model Repository Approach

The CViT DMR provides a SOAP/WSDL Web Service based on the National Cancer Institute's cancer Biomedical Informatics Grid (caBIG) specification.

- The **advantage** is that the DMR caBIG Service can utilize the caBIG infrastructure (caDSR - repository of UML models for all caBIG services; caGrid Portal - UDDI-like index of all running caBIG services; EVS - enterprise vocabulary service for consistent use of terminology; and Grid Trust Service/Dorian - for user authentication and delegation of trust across services). All caBIG Data Services implement a simple caBIG Query Language (CQL), based on XML, 'query' interface to locate and retrieve any data exposed by the UML model (stored in caDSR). This makes it easy for any 'novice' user to find available caBIG services, understand what the data means, and query/retrieve appropriate data.
- The **disadvantage** is that the caBIG infrastructure has a lot of overhead (Globus Toolkit, Dorian), which is not trivial to install and configure correctly. More troubling is that all of the packages (Globus, Dorian, caBIG) depend on different versions of the same Java library (which is poorly documented), which requires you to 'guess-and-test' various versions of the libraries until you have a working system.

CViT's DMR communicates with the Computational Model Execution Framework (CMEF) using a REST interface. This is a much lighter, easier to implement interface; however, it does not have the many useful services provided by caBIG (Grid Trust, Security, caDSR, EVS/Semantics). To prevent unauthorized use, the CMEF runs on a private network and only allows computational jobs to be submitted through the DMR's Website. Unlike the CViT DMR caBIG Data Service, the CViT CMEF service is not semantically annotated.

REST may be easier to implement, but including REST Security is a challenge: (http://www.infoq.com/news/2010/03/REST_security, http://features.techworld.com/security/3213655/the-security-nightmare-of-rest-web-services/, http://features.techworld.com/security/3213655/the-security-nightmare-of-rest-web-services/) Also, the less structured interface may require more/better end-user documentation (e.g., API specs, how to configure security, etc.) to properly use the service. It should be possible to semantically annotate REST results; however, there is no standard for specifying REST semantics.

The more structured API of SOAP/WSDL may make it easier for client programs to utilize the Web Service, including using built-in WS-Security. Workflow tools such as Taverna (http://www.taverna.org.uk/developers/web-service-developers/) support including SOAP/WSDL services. Many development tools support client code generation from WSDL specs.

--Thomas T. 15:55, 16 February 2011 (EST)

Brief Survey of Selected Web Services on the Internet

http://www.programmableweb.com

Name	Description	REST	SOAP
CViT DMR caBIG Data Service	Through CViT's Digital Model Repository (DMR), scientists can add new models, share models with other researchers, and discuss model simulations. The website also allows for starting grid-based execution of the models in the repository. eLicensing Workflow protects intellectual property . The CViT DMR caBIG Data Service is a caBIG Silver-Level Compliant Web Service that provides secure access to the DMR through caGrid.		Yes
CViT CMEF	Through CViT's Computational Model Execution Framework (CMEF), scientists can perform grid-based execution of the models stored in the DMR. CMEF Server runs Java, C, and R programs on both Windows and Linux, and models are automatically annotated with execution metadata.	Yes	
NCI Cancer Biomedical Informatics Grid	The mission of caBIG® is to develop a collaborative information network that accelerates the discovery of new approaches for the detection, diagnosis, treatment, and prevention of cancer. caBIG® is sponsored by the National Cancer Institute (NCI) and its activities are supervised by the National Cancer Institute Center for Bioinformatics and Information Technology (NCI-CBIIT). The initiative operates through an open development community.		Yes
Taverna API	Taverna is an open source domain independent Workflow Management System – a suite of tools used to design and execute scientific workflows. Taverna has been created by the myGrid project and funded through the OMII-UK. Taverna has guaranteed funding till 2014.		Yes
KEGG API	KEGG API provides valuable means for accessing the KEGG system, such as for searching and computing biochemical pathways in cellular processes or analyzing the universe of genes in the completely sequenced genomes. This enables the users to write their own programs for many different purposes and to automate the procedure of accessing the KEGG API server and retrieving the results.		Yes
Reactome Data Sharing API	A SOAP based Web Services API is available to access the Reactome data. Reactome is a human-curated knowledgebase of biological pathways and reactions. The information in this knowledgebase is authored by biological researchers with expertise in their fields, maintained by the Reactome editorial staff, and cross-		Yes

	referenced with PubMed, Gene Ontology, NCBI, Ensembl, UniProt, OMIM and other databases.		
PSICQUIC Standard	Proteomics Standard Initiative Common Query Interface. PSICQUIC is an effort from the HUPO Proteomics Standard Initiative (HUPO-PSI) to standardize the access to molecular interaction databases programmatically. Several interaction databases are already implementing PSICQUIC: IntAct, MINT, MPIDB, MatrixDB, BioGrid, iRefIndex, InnateDB, APID, STRING and ChEMBL.	Yes	Yes
IntAct	IntAct is the Molecular Interaction database at the European Bioinformatics Institute (UK). Implements PSICQUIC.	Yes	Yes
MINT	MINT, the Molecular INTeraction database. MINT focuses on experimentally verified protein-protein interactions mined from the scientific literature by expert curators. Implements PSICQUIC.	Yes	Yes
Pathway Commons API	Pathway Commons is a convenient point of access to biological pathway information collected from public pathway databases, which you can browse or search. You can programmatically access pathway data via the Web Service API.	HTTP/URL	
Amazon eCommerce API	What was formerly the ECS - eCommerce Service - has been renamed the Product Advertising API. Through this API developers can retrieve product information. The API exposes Amazon's product data and e-commerce functionality.	Yes	Yes
Amazon S3 API	Amazon S3 API, the Simple Storage Service provides a simple web services interface used to store objects using the Amazon online storage infrastructure.	Yes	Yes
Business.gov API	Business.gov is an official site of the US government that helps small businesses understand their legal requirements, and locate government services supporting the nation's small business community.	Yes	
CareerBuilder API	CareerBuilder API allows you to integrate their database of over 1.5 million jobs with your software or website. The API provides methods for basic job searching, retrieving data for a specific job, and applying to a job.	Yes	
Deutsche Telekom Voice Call API	The Deutsche Telekom Voice Call API can be used to initiate voice calls between phones.		Yes

Dun and Bradstreet Credit Check API	Perform low risk credit assessments and pre-screen prospects with D&B's core credit evaluation data. Information includes company identification, payment activity summary, public filings indicators, and the D&B Rating.		Yes
E*TRADE API	With E*TRADE's open XML-based Application Programming Interface, you can now use industry-leading trading applications to execute your trades directly through E*TRADE - no additional software needed.	Yes	
eBay API	World's largest online auction service. API allows for both searching of products and upload of new listings.	Yes	Yes
Facebook API	The Facebook API is a platform for building applications that are available to the members of the social network of Facebook. The API allows applications to use the social connections and profile information to make applications more involving, and to publish activities to the news feed and profile pages of Facebook, subject to individual users privacy settings.	Yes	
Google Ajax Search API	The Google AJAX Search API is a Javascript library that allows you to embed Google Search in your web pages and other web applications. You can embed a simple, dynamic search box and display search results in your own web pages or use the results in innovative, programmatic ways.	Yes	
Google AdSense API	Using the AdSense API, developers can let users sign up for AdSense through their site or program, generate detailed performance reports for users, and choose how the AdSense revenue is shared with our revenue sharing program.		Yes
Google Base API	The Google Base API makes it possible to get items in and out of Google Base; a free service for submitting various content for Google to host and to make searchable online. The Google Base data API is designed to enable developers to do two things: (1) Query Google Base data to create applications and mashups. (2) Input and manage Google Base items programmatically.	Yes	
Microsoft MSDN API	The MTPS Content Service is an XML web service that provides access to the content stored in MTPS. Using the MTPS Content Service, developers can integrate documentation, technical articles, whitepapers, images and the other content available from the MTPS system into their own applications.		Yes

Microsoft HealthVault API	Use the HealthVault API to access an open platform for giving your users data storage, security features, and authentication. Use the platform to make it easier to collect health data from other apps, devices, lab tests, etc., and share the data with HealthVault-compatible apps.	Yes	
Scopus API	Search the largest abstract and citation database of peer-reviewed literature and quality web sources. Include Scopus content in your website, perhaps to broadcast your own organization's researchers. You can access more than 33 million continuously updated records.	JavaScript	
SciVerse Framework and Content API	With SciVerse APIs, developers can build applications that appear alongside full text articles, journal abstracts and search results within the SciVerse Product Suite: ScienceDirect, Scopus and SciVerse Hub.	Yes	
Swoogle API	Use the Swoogle API to search semantic web ontologies, markup and documents hosted by UMBC. The system is the result of ongoing research. It provides a set of functions, such as search Semantic Web ontologies, search Semantic Web instance data, etc.	Yes	

3 Review of Web Services Technologies

A "Web service" is a method of communication between two electronic devices over a network. The W3C describes a Web service as "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards." (http://www.w3.org/TR/ws-gloss/)

W3C describes the set of interrelated technologies that can be utilized to construct and consume Web services, as illustrated in Figure 1.

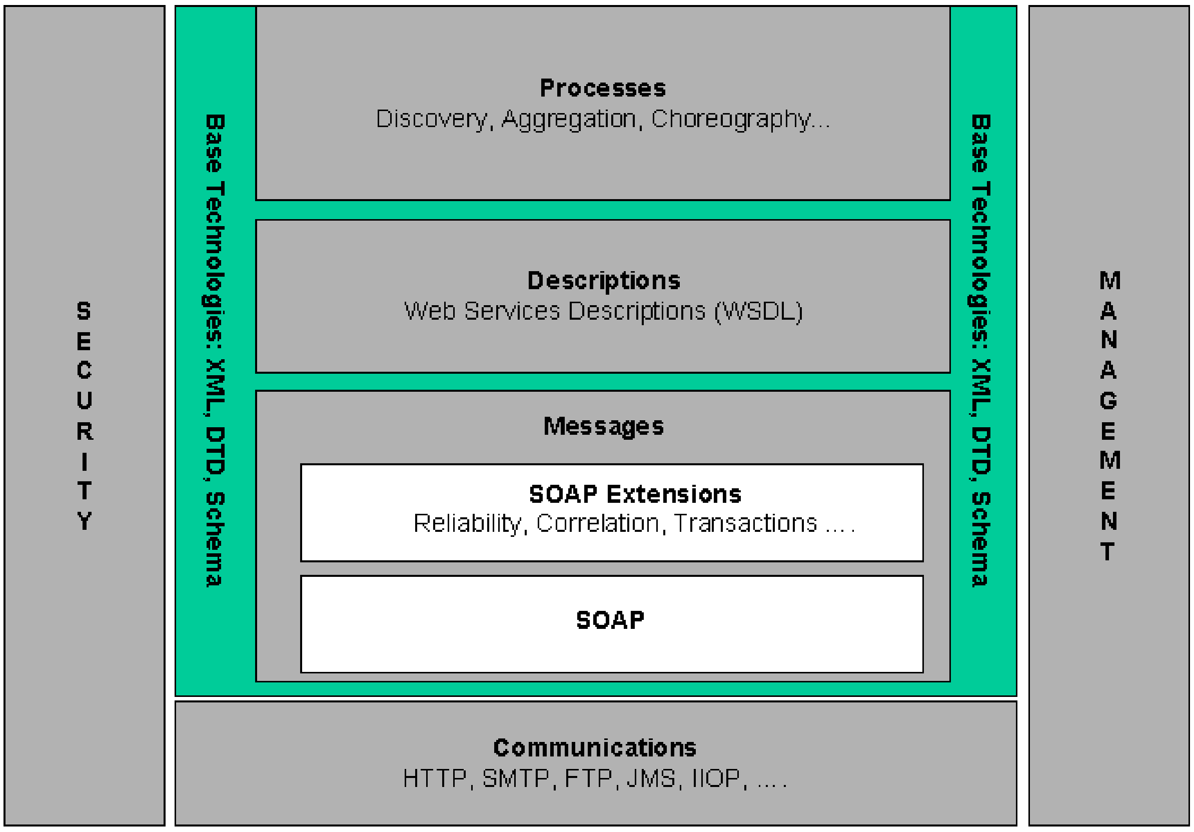

SECURITY
Base Technologies: XML, DTD, Schema
Processes
Discovery, Aggregation, Choreography...
Descriptions
Web Services Descriptions (WSDL)
Messages
SOAP Extensions
Reliability, Correlation, Transactions ....
SOAP
Base Technologies: XML, DTD, Schema
MANAGEMENT
Communications
HTTP, SMTP, FTP, JMS, IIOP, ...

Figure 1. Web Services Architecture Stack from http://www.w3.org/TR/ws-arch

- **XML** – provides a standard, flexible, and extensible data format that reduces the burden of deploying the technologies utilized in Web services. Key concepts are XML core syntax, XML Infoset, XML Schema, and XML Namespaces.
- **SOAP** – provides a standard, extensible, composable framework for packaging and exchanging XML messages. SOAP also provides a mechanism for referencing capabilities (a named feature or piece of functionality that is declared or requested by an agent).
  - *SOAP 1.2 Part 1* – defines an XML-based messaging framework, a processing model, and an extensibility model.
  - *SOAP 1.2 Part 2* – defines encoding rules for expressing instances of application-defined data types, conventions for representing remote procedure calls and responses, and rules for using SOAP with HTTP.

- **WSDL –** defines a language for describing Web services. WSDL describes the messages that are exchanged between requester and provider agents. The messages are described abstractly and bound to a concrete network protocol and message format. Web service definitions can be mapped to any implementation language, platform, object model, or messaging system.

Web Service styles of use

Web services are a set of tools that can be used in a number of ways. The three most common styles of use are RPC, SOA and REST (http://en.wikipedia.org/wiki/Web_service).

- **Remote procedure calls -** RPC Web services present a distributed function (or method) call interface that is familiar to many developers. Typically, the basic unit of RPC Web services is the WSDL operation. Other approaches with nearly the same functionality as RPC are Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA), Microsoft's Distributed Component Object Model (DCOM) or Sun Microsystems's Java/Remote Method Invocation (RMI).
- **Service-oriented architecture** - Web services can be implemented according to service-oriented architecture (SOA) concepts, where the basic unit of communication is a message, rather than an operation. Unlike RPC Web services, loose coupling is more likely, because the focus is on the "contract" that WSDL provides, rather than the underlying implementation details.
- **Representational state transfer (REST) -** REST attempts to describe architectures that use HTTP or similar protocols by constraining the interface to a set of well-known, standard operations (like GET, POST, PUT, DELETE for HTTP). Here, the focus is on interacting with stateful resources, rather than messages or operations. An architecture based on REST can use WSDL to describe SOAP messaging over HTTP, can be implemented as an abstraction purely on top of SOAP, or can be created without using SOAP at all.

Web Services Description Language (WSDL)

Web Services Description Language (WSDL) Version 2.0 Part 0: Primer

W3C Recommendation 26 June 2007

http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/

Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language

W3C Recommendation 26 June 2007

http://www.w3.org/TR/2007/REC-wsdl20-20070626/

Web Services Description Language (WSDL) provides a model and an XML format for describing Web services. WSDL 2.0 enables one to separate the description of the abstract functionality offered by a service from concrete details of a service description such as “how” and “where” that functionality is offered. The WSDL 2.0 specification defines a language for describing the abstract functionality of a service as well as a framework for describing the concrete details of a service description. It also defines the conformance criteria for documents in this language.

3.1.1 Service Description

A service description indicates how potential clients are intended to interact with the described service. It represents an assertion that the described service fully implements and conforms to what the WSDL document describes. WSDL describes a Web service in two fundamental stages: one abstract and one concrete. At an abstract level, WSDL describes a Web service in terms of the messages it sends and receives; messages are described independent of a specific wire format using a type system, typically XML Schema.

An operation associates a message exchange pattern with one or more messages. A message exchange pattern identifies the sequence and cardinality of messages sent and/or received as well as who they are logically sent to and/or received from. An interface groups together operations without any commitment to transport or wire format.

At a concrete level, a binding specifies transport and wire format details for one or more interfaces. An endpoint associates a network address with a binding. And finally, a service groups together endpoints that implement a common interface.

3.1.2 Component Model

Components are typed collections of properties that correspond to different aspects of Web services.

- **Description** - a container for two categories of components: WSDL 2.0 components and type system components. WSDL 2.0 components are interfaces, bindings and services. Type system components are element declarations and type definitions. An Element Declaration component defines the name and content model of an element information item such as that defined by an XML Schema global element declaration. A Type Definition component defines the content model of an element information item such as that defined by an XML

Schema global type definition. Interface, Binding, Service, Element Declaration, and Type Definition components are directly contained in the Description component and are referred to as top-level components.

```
<description
      targetNamespace="xs:anyURI" >
  <documentation />*
  [ <import /> | <include /> ]*
  <types />?
  [ <interface /> | <binding /> | <service /> ]*
</description>
```

- **Interface** - describes sequences of messages that a service sends and/or receives. It does this by grouping related messages into operations. An operation is a sequence of input and output messages, and an interface is a set of operations. The set of operations available in an interface includes all the operations defined by the interfaces it extends directly or indirectly, together with any operations it directly defines.

```
<description>
  <interface
        name="xs:NCName"
        extends="list of xs:QName"?
        styleDefault="list of xs:anyURI"? >
    <documentation />*
    [ <fault /> | <operation /> ]*
  </interface>
</description>
```

- **Interface Fault** - describes a fault that may occur during invocation of an operation of the interface. A fault is an event that occurs during the execution of a message exchange that disrupts the normal flow of messages. A fault is typically raised when a party is unable to communicate an error condition inside the normal message flow, or a party wishes to terminate a message exchange. A fault message may be used to communicate out of band information such as the reason for the error, the origin of the fault, as well as other informal diagnostics such as a program stack trace.

```
<description>
  <interface>
    <fault
          name="xs:NCName"
          element="union of xs:QName, xs:token"? >
      <documentation />*
    </fault>
  </interface>
</description>
```

- **Interface Operation** - describes an operation that a given interface supports. An operation is an interaction with the service consisting of a set of messages exchanged between the service and the other parties involved in the interaction. A message exchange pattern defines placeholders for messages, the participants in the pattern, and the cardinality and sequencing of messages exchanged by the participants. The message placeholders are associated with specific message types by the operation that uses the pattern by means of message and fault references.

```
<description>
  <interface>
    <operation
          name="xs:NCName"
          pattern="xs:anyURI"?
          style="list of xs:anyURI"? >
      <documentation />*
      [ <input /> | <output /> | <infault /> | <outfault /> ]*
    </operation>
  </interface>
</description>
```

- **Interface Message Reference** - defines the content, or payload, of a message exchanged in an operation. By default, the message content is defined by an XML-based type system such as XML Schema. Other type systems may be used via the WSDL type system extension mechanism. A message exchange pattern defines a set of placeholder messages that participate in the pattern and assigns them unique message labels within the pattern.

```
<description>
  <interface>
    <operation>
      <input
            messageLabel="xs:NCName"?
            element="union of xs:QName, xs:token"? >
        <documentation />*
      </input>
      <output
            messageLabel="xs:NCName"?
            element="union of xs:QName, xs:token"? >
        <documentation />*
      </output>
    </operation>
  </interface>
</description>
```

- **Interface Fault Reference** - associates a defined type, specified by an Interface Fault component, to a fault message exchanged in an operation. The purpose of an Interface Fault Reference component is to associate an actual message type with a fault message occurring in the pattern.

```
<description>
  <interface>
    <operation>
      <infault
            ref="xs:QName"
            messageLabel="xs:NCName"? >
        <documentation />*
```

```
       </infault>*
       <outfault
             ref="xs:QName"
             messageLabel="xs:NCName"? >
         <documentation />*
       </outfault>*
     </operation>
   </interface>
</description>
```

- **Binding** - describes a concrete message format and transmission protocol which may be used to define the implementation details necessary to access the service. Binding information may be specified on a per-operation basis within an interface, in addition to across all operations of an interface. Binding components that do not specify an interface may be used to specify operation-independent binding details for Service components with different interfaces.

```
<description>
  <binding
        name="xs:NCName"
        interface="xs:QName"?
        type="xs:anyURI" >
    <documentation />*
    [ <fault /> | <operation /> ]*
  </binding>
</description>
```

- **Binding Fault** – describes a concrete binding of a particular fault within an interface to a particular concrete message format. The fault binding information specified in a Binding Fault component describes how faults that occur within a message exchange of an operation will be formatted and carried in the transport.

```
<description>
  <binding>
    <fault
         ref="xs:QName" >
      <documentation />*
    </fault>
  </binding>
</description>
```

- **Binding Operation** - describes the concrete message format(s) and protocol interactions associated with a particular interface operation for a given endpoint.

```
<description>
  <binding>
    <operation
          ref="xs:QName" >
      <documentation />*
      [ <input /> | <output /> | <infault /> | <outfault /> ]*
    </operation>
  </binding>
</description>
```

- **Binding Message Reference -** describes a concrete binding of a particular message participating in an operation to a particular concrete message format.

```
<description>
  <binding>
    <operation>
      <input
            messageLabel="xs:NCName"? >
        <documentation />*
      </input>
      <output
            messageLabel="xs:NCName"? >
        <documentation />*
      </output>
    </operation>
  </binding>
</description>
```

- **Binding Fault Reference** - describes a concrete binding of a particular fault participating in an operation to a particular concrete message format.

```
<description>
  <binding>
    <operation>
      <infault
            ref="xs:QName"
            messageLabel="xs:NCName"?>
        <documentation />*
      </infault>
      <outfault
            ref="xs:QName"
            messageLabel="xs:NCName"?>
        <documentation />*
      </outfault>
    </operation>
  </binding>
</description>
```

- **Service** - describes a set of endpoints at which a particular deployed implementation of the service is provided. The endpoints are in effect alternate places at which the service is provided.

```
<description>
  <service
        name="xs:NCName"
        interface="xs:QName" >
    <documentation />*
    <endpoint />+
  </service>
</description>
```

- **Endpoint** - defines the particulars of a specific endpoint at which a given service is available. Endpoint components are local to a given Service component. The Binding component is said to be applied to the Interface component which is the value of the interface property of the parent Service component of the Endpoint.

```
<description>
  <service>
    <endpoint
          name="xs:NCName"
          binding="xs:QName"
          address="xs:anyURI"? >
      <documentation />*
    </endpoint>+
  </service>
</description>
```

3.1.3 Types

The content of messages and faults may be constrained using type system components. These constraints are based upon a specific data model, and expressed using a particular schema language. WSDL only defines a means of expressing constraints based upon the XML Infoset, and WSDL only defines the use of XML Schema. Specifically, the element declarations and type definitions properties of the description component are collections of imported and inlined schema components that describe Infoset element information items. The schema components contained in the element declarations property of the Description component provide the type system used for Interface Message Reference and Interface Fault components. Interface Message Reference components indicate their structure and content by using the standard attribute information items element. Interface Fault components behave similarly.

```
<description>
  <types>
    <documentation />*
    [ <xs:import namespace="xs:anyURI" schemaLocation="xs:anyURI"? /> |
      <xs:schema targetNamespace="xs:anyURI"? /> |
      other extension elements ]*
  </types>
</description>
```

Simple Object Access Protocol / Service Oriented Architecture Protocol (SOAP)

SOAP Version 1.2 Part 0: Primer (Second Edition)

W3C Recommendation 27 April 2007

http://www.w3.org/TR/2007/REC-soap12-part0-20070427/

SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)

W3C Recommendation 27 April 2007

http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

SOAP Version 1.2 is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. "Part 1: Messaging Framework" defines, using XML technologies, an extensible messaging framework containing a message construct that can be exchanged over a variety of underlying protocols. SOAP is fundamentally a stateless, one-way message exchange paradigm, but applications can create more complex interaction patterns (e.g., request/response, request/multiple responses, etc.) by combining such one-way exchanges with features provided by an underlying protocol and/or application-specific information. SOAP is silent on the semantics of any application-specific data it conveys, as it is on issues such as the routing of SOAP messages, reliable data transfer, firewall traversal, etc. However, SOAP provides the framework by which application-specific information may be conveyed in an extensible manner.

3.1.4 SOAP Messages

A SOAP message is specified as an XML infoset whose comment, element, attribute, namespace and character information items are able to be serialized as XML 1.0.

- **SOAP Envelope** - The SOAP Envelope element information item has: A [local name] of Envelope; A [namespace name] of http://www.w3.org/2003/05/soap-envelope; Zero or more namespace-qualified attribute information items amongst its [attributes] property; One or two element information items in its [children] property in order as follows: An optional Header element information item; A mandatory Body element information item. (See Figure 2)

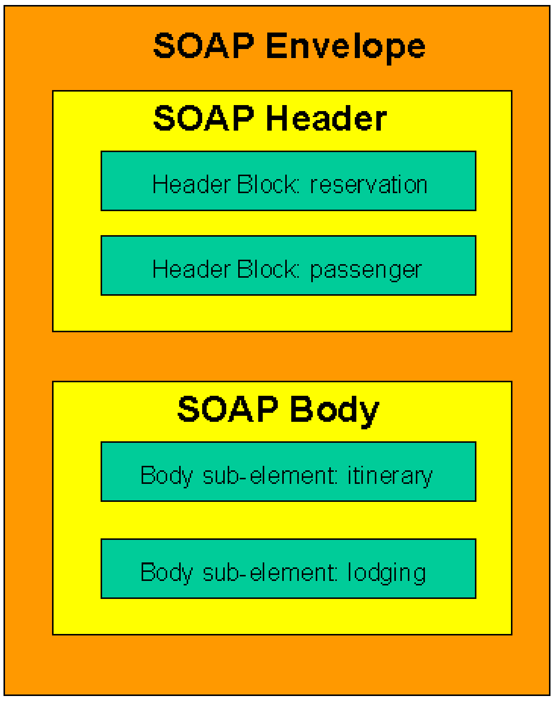

SOAP Envelope
SOAP Header
Header Block: reservation
Header Block: passenger
SOAP Body
Body sub-element: itinerary
Body sub-element: lodging

Figure 2. SOAP Message Structure from http://www.w3.org/TR/2007/REC-soap12-part0

- **SOAP Header** - The SOAP Header element information item provides a mechanism for extending a SOAP message in a decentralized and modular way. The SOAP Header element is optional. A SOAP header is an extension mechanism that provides a way to pass information in SOAP messages that is not application payload. Such "control" information includes, for example, passing directives or contextual information related to the processing of the message. This allows a SOAP message to be extended in an application-specific manner.

- **SOAP Body** - A SOAP body provides a mechanism for transmitting information to an ultimate SOAP receiver. The SOAP body is the mandatory element within the SOAP env:Envelope, which implies that this is where the main end-to-end information conveyed in a SOAP message must be carried.

- **SOAP Fault -** A SOAP fault is used to carry error information within a SOAP message. To be recognized as carrying SOAP error information, a SOAP message MUST contain a single SOAP Fault element information item as the only child element information item of the SOAP Body.

3.1.5 SOAP - Remote Procedure Calls

One of the design goals of SOAP Version 1.2 is to encapsulate remote procedure call functionality using the extensibility and flexibility of XML. To invoke a SOAP RPC, the following information is needed:

- The address of the target SOAP node.
- The procedure or method name.
- The identities and values of any arguments to be passed to the procedure or method together with any output parameters and return value.
- A clear separation of the arguments used to identify the Web resource which is the actual target for the RPC, as contrasted with those that convey data or control information used for processing the call by the target resource.
- The message exchange pattern which will be employed to convey the RPC, together with an identification of the so-called "Web Method" (on which more later) to be used.
- Optionally, data which may be carried as a part of SOAP header blocks.

Semantic Annotations for WSDL and XML Schema (SAWSDL)

W3C Recommendation 28 August 2007

http://www.w3.org/TR/2007/REC-sawsdl-20070828/

The W3C recommendation Semantic Annotations for WSDL and XML Schemas (SAWSDL) defines a set of extension attributes for WSDL and XML Schema providing for incorporating semantics. SAWSDL describes how semantic annotation is accomplished using ontologies and provides an annotation mechanism where concepts from ontologies can be referenced from within WSDL and XML schema.

3.1.6 Annotation Mechanisms

WSDL utilizes five components to describe service definitions. Element Declaration, Type Definition, and Interface deal with the abstract definition of a service. Binding and Service deal with service implementation. SAWSDL focuses on semantically annotating the abstract definition of a service to support dynamic discovery, composition, and invocation of services. SAWSDL defines three extension attributes:

- Model Reference - specifies the association between a WSDL or XML Schema component and a concept in some semantic model. It is used to annotate XML Schema type definitions, element declarations, and attribute declarations as well as WSDL interfaces, operations, and faults. The modelReference attribute is a set of zero or more URIs, separated by whitespaces, that identify concepts in a semantic model. Each URI is a pointer to a concept in a semantic model and is intended to provide semantic information about the WSDL or XML Schema component being annotated.
- Schema Mapping (liftingSchemaMapping, loweringSchemaMapping) - are added to XML Schema element declarations and type definitions for specifying mappings between semantic data and XML. Schema mapping relates the instance data defined by an XML Schema document with some semantic data defined by a semantic model. Schema mapping annotations address post-discovery issues in using a Web service. In general, lifting schema mappings lift data from XML to a semantic model, whereas lowering schema mappings lower data from a semantic model into an XML structure.

3.1.7 Annotating WSDL

A model reference is a new property for WSDL. The modelReference annotation on xs:element, xs:complexType, xs:simpleType and xs:attribute defines the semantics of the input or output data of WSDL operations. A modelReference on a WSDL operation or fault gives semantic information about that operation, while a modelReference on a WSDL interface provides a classification or other semantic descriptions of the interface.

- **Interfaces** – A modelReference on a WSDL interface element provides a reference to a concept or concepts in a semantic model that describe the Interface. For taxonomies whose elements are not identifiable with a URI, the modelReference can point to a simple semantic model that contains the taxonomy reference information. SAWSDL does not constrain the form of the semantic model for categorization or that of any other semantic model specified in a modelReference on an interface.

```
...
<wsdl:interface                                          name=“Order”
sawsdl:modelReference=“http://example.org/categorization/products/el
ectronics”>
  ...
</wsdl:interface>
...
```

- **Operations** – The annotation of an operation element carries a reference to a concept in a semantic model that provides a high level description of the operation, specifies its behavioral aspects or includes other semantic definitions. Although inputs and outputs provide one way of capturing the semantics of an operation, a simple semantic annotation indicating the intended behavior of a given operation as a verb concept may be useful at certain times. During service discovery, this verb provides a coarse indication of whether this service is a match for a given request. Operations can also be annotated with category references. These are separate from any categorizations applied to other WSDL components.

```
...
<wsdl:operation  name=”order”  pattern=”http://www.w3.org/ns/wsdl/in-
out”

sawsdl:modelReference=”http://www.w3.org/2002/ws/sawsdl/spec/ontology
/purchaseorder#RequestPurchaseOrder”>
  <wsdl:input element=”OrderRequest”/>
  <wsdl:output element=”OrderResponse”/>
</wsdl:operation>
...
```

- **Faults** – The annotation of the fault element carries a reference to a concept in a semantic model that provides a high level description of the fault and can include other semantic definitions. The fault annotation does not describe the fault message, which should be annotated in the XML schema.

```
...
<wsdl:interface name=”Order”>


  <wsdl:fault                               name=”ItemUnavailableFault”
element=”AvailabilityInformation”

sawsdl:modelReference=”http://www.w3.org/2002/ws/sawsdl/spec/ontology/p
urchaseorder#ItemUnavailable”/>
  ...
</wsdl:interface>
```

3.1.8 Annotating XML Schema

SAWSDL supports both model references and schema mappings to annotate XML Schema (see Annotation Mechanisms).

- **Simple Types -** Simple types can be annotated by including a modelReference attribute on the xs:simpleType element.

```
...
<xs:simpleType name="Confirmation"

sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/pur
chaseorder#OrderConfirmation">
  ...
</xs:simpleType>
...
```

- **Complex Types –** Complex types can be annotated using Bottom Level Annotation (at member element or attribute level) or Top Level Annotation (at complex type container level). In bottom level annotation, all the member elements and attributes in a complex type can be annotated. In top level annotation, the complex types themselves are annotated with model references. If multiple concepts describe the complex type, all of their URIs can be included in the value of the modelReference attribute.

Bottom Level Annotation

```
...
<xs:complexType>
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element name="quantity" type="xs:integer"

sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/pur
chaseorder#Quantity"/>
    <xs:element name="UPC" type="xs:string"

sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/pur
chaseorder#ProductCode"/>
  </xs:sequence>
</xs:complexType>
...
```

Top Level Annotation

```
...
<xs:complexType
sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/purch
aseorder#OrderRequest">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element name="quantity" type="xs:integer"
    <xs:element name="UPC" type="xs:string"
  </xs:sequence>

</xs:complexType>

...
```

- **Elements –** An element declaration can be annotated by including a modelReference on the xs:element element.

```
...

<xs:element name="OrderRequest"


sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/purch
aseorder#OrderRequest">

  <xs:complexType>

    <xs:sequence>

      <xs:element name="customerNo" type="xs:integer" />

      <xs:element      name="orderItem"      type="item"      minOccurs="1"
maxOccurs="unbounded" />

    </xs:sequence>

  </xs:complexType>

</xs:element>

...
```

- **Attributes** - An attribute can be annotated by including a modelReference on the xs:attribute element.

```
...

<xs:attribute name="quantity" type="xs:integer"


sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/purch
aseorder#Quantity"/>

...
```

- **Schema Mapping** - liftingSchemaMapping and loweringSchemaMapping are used to associate a schema type or element with a mapping to an ontology. Schema mappings may be added to global type definitions (complex or simple) as well as to global element declarations. A mapping referenced by liftingSchemaMapping defines how an XML instance document conforming to the element or type defined in a schema is transformed to data that conforms to some semantic model, i.e. the output of the transformation process will be semantic data. A mapping referenced by loweringSchemaMapping defines how data in a semantic model is transformed to XML instance data.

Lifting Schema Mapping

```
...

<xs:element name="OrderResponse" type="confirmation" />

<xs:simpleType name="confirmation"
```

```
sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/purch
aseorder#OrderConfirmation"

sawsdl:liftingSchemaMapping="http://www.w3.org/2002/ws/sawsdl/spec/mapping/
Response2Ont.xslt">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Confirmed" />
    <xs:enumeration value="Pending" />
    <xs:enumeration value="Rejected" />
  </xs:restriction>
</xs:simpleType>
...
```

Representational State Transfer (REST) Web Services

Architectural Styles and the Design of Network-based Software Architectures

Roy Fielding Ph.D. Dissertation, University of California, Irvine, 2000

http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

The Representational State Transfer (REST) style is an abstraction of architectural elements within a distributed hypermedia system. REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements. It encompasses the fundamental constraints upon components, connectors, and data that define the basis of the Web architecture, and thus the essence of its behavior as a network-based application.

REST-style architectures consist of clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of representations of resources. A resource can be essentially any coherent and meaningful concept that may be addressed. A representation of a resource is typically a document that captures the current or intended state of a resource. The client begins sending requests when it is ready to make the transition to a new state. While one or more requests are outstanding, the client is considered to be in transition. The representation of each application state contains links that may be used next time the client chooses to initiate a new state transition.

In HTTP(S) based RESTful web services, the emphasis is on simple point-to-point communication over HTTP using XML. REST is a hybrid style derived from several of the network-based architectural styles and combined with additional constraints that define a uniform connector interface. REST architectures that use the HTTP application protocol can be summed up as using five verbs (GET, HEAD, POST, PUT, and DELETE methods from HTTP 1.1) and the nouns, which are the resources available on the network (referenced in the URI). The verbs have the following operational equivalents:

HTTP	**CRUD Equivalent**	**Safe**	**Idempotent**
GET	Read	☑	☑
HEAD	*Get metadata*	☑	☑
POST	Create, Update, Delete		
PUT	Create, Update		☑
DELETE	Delete		☑

In the REST “circles” the convention has been established that the GET and HEAD methods should not have the significance of taking an action other than retrieval i.e. they don't have side effects. These methods ought to be considered "safe". Methods can also have the property of "idempotence" in that (aside from error or expiration issues) the side-effects of two or more identical requests is the same as for a single request. The methods GET, HEAD, PUT and DELETE share this property.

3.1.9 Data Elements

REST data elements focus on a shared understanding of data types with metadata, but limit the scope of what is revealed to a standardized interface. REST components communicate by transferring a representation of a resource in a format matching one of an evolving set of standard data types, selected dynamically based on the capabilities or desires of the recipient and the nature of the resource.

Data Element	Modern Web Examples
resource	the intended conceptual target of a hypertext reference
resource identifier	URL, URN
representation	HTML document, JPEG image
representation metadata	media type, last-modified time
resource metadata	source link, alternates, vary
control data	if-modified-since, cache-control

Figure 3. REST Data Elements from http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

3.1.10 Resources and Resource Identifiers

The key abstraction of information in REST is a resource. Any information that can be named can be a resource. A resource is a conceptual mapping to a set of entities, not the entity that corresponds to the mapping at any particular point in time. REST uses a resource identifier to identify the particular resource involved in an interaction between components. REST connectors provide a generic interface for accessing and manipulating the value set of a resource, regardless of how the membership function is defined or the type of software that is handling the request. The naming authority that assigned the resource identifier, making it possible to reference the resource, is responsible for maintaining the semantic validity of the mapping over time.

3.1.11 Representations

REST components perform actions on a resource by using a representation to capture the current or intended state of that resource and transferring that representation between components. A representation consists of data, metadata describing the data, and, on occasion, metadata to describe the metadata (usually for the purpose of verifying message integrity). Metadata is in the form of name-value pairs, where the name corresponds to a standard that defines the value's structure and semantics. Response messages may include both representation metadata and resource metadata: information about the resource that is not specific to the supplied representation. Control data defines the purpose of a message between components, such as the action being requested or the meaning of a response. It is also used to parameterize requests and override the default behavior of some connecting elements.

3.1.12 Connectors

REST uses various connector types to encapsulate the activities of accessing resources and transferring resource representations. The connectors present an abstract interface for component communication, enhancing simplicity by providing a clean separation of concerns and hiding the underlying implementation of resources and communication mechanisms.

All REST interactions are stateless. This restriction accomplishes four functions: 1) it removes any need for the connectors to retain application state between requests, thus reducing consumption of physical resources and improving scalability; 2) it allows interactions to be processed in parallel without requiring that the processing mechanism understand the interaction semantics; 3) it allows an intermediary to view and understand a request in isolation, which may be necessary when services are dynamically rearranged; and, 4) it forces all of the information that might factor into the reusability of a cached response to be present in each request.

Connector	Modern Web Examples
client	Libwww, libwww-perl, Apache HttpComponents (java)
server	Libwww, Apache API, NSAPI, ISAPI, WSGI (Python)
cache	Browser cache, Web Proxy servers, Akamai cache network
resolver	Bind (DNS lookup library)
tunnel	SOCKS, SSL after HTTP CONNECT

Figure 4. REST Connectors from http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

3.1.13 Components

REST components are typed by their roles in an overall application action. A user agent uses a client connector to initiate a request and becomes the ultimate recipient of the response. An origin server uses a server connector to govern the namespace for a requested resource. It is the definitive source for representations of its resources and must be the ultimate recipient of any request that intends to modify the value of its resources. Intermediary components act as both a client and a server in order to forward, with possible translation, requests and responses.

Component	Modern Web Examples
origin server	Apache httpd, Microsoft IIS
gateway	Squid, CGI, Reverse Proxy
proxy	CERN Proxy, Netscape Proxy, Gauntlet
user agent	Browsers (e.g. Firefox), “search bots”/“spiders” (e.g. GoogleBot)

Figure 5. REST Components from http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Web Application Description Language (WADL)

Web Application Description Language

W3C Member Submission 31 August 2009

http://www.w3.org/Submission/wadl/

An increasing number of Web-based enterprises are developing HTTP-based applications that provide programmatic access to their internal data. Typically these applications are described using textual documentation that is sometimes supplemented with more formal specifications such as XML schema for XML-based data formats. WADL is designed to provide a machine process-able description of such HTTP-based Web applications. Oracle, formerly Sun Microsystems, has submitted the Web Application Description Language (WADL) to the W3C for consideration.

WADL provides mechanisms for describing Web applications in a machine processable format, this includes:

- **Set of resources -** Analogous to a site map showing the resources on offer.
- **Relationships between resources -** Describing the links between resources, both referential and causal.
- **Methods that can be applied to each resource -** The HTTP methods that can be applied to each resource, the expected inputs and outputs and their supported formats.
- **Resource representation formats -** The supported MIME types and data schemas in use.

WADL is intended to provide a common foundation for individual applications and protocols to reuse and extend. WADL highlights several use cases not supported by current state-of-the-art Web application description:

- **Application Modeling and Visualization -** Support for development of resource modeling tools for resource relationship and choreography analysis and manipulation.
- **Code Generation -** Automated generation of stub and skeleton code and code for manipulation of resource representations.
- **Configuration -** Configuration of client and server using a portable format.

WADL defines the following components to describe a Web application:

- **Application** – The `application` element forms the root of a WADL description
- **Documentation** - Each WADL-defined element can have one or more child `doc` elements that can be used to document that element.
- **Grammars** – The `grammars` element acts as a container for definitions of the format of data exchanged during execution of the protocol described by the WADL document. Such definitions may be included inline or by reference using the `include` element.
- **Resources** – The resources element acts as a container for the resources provided by the application.
- **Resource Type** - A `resource_type` element describes a set of methods that, together, define the behavior of a type of resource. A `resource_type` may be used to define resource behavior that is expected to be supported by multiple resources.

- **Method** - A `method` element describes the input to and output from an HTTP protocol method that may be applied to a resource. A `method` element can either be a method definition or a reference to a method defined elsewhere.
- **Request** - A `request` element describes the input to be included when applying an HTTP method to a resource.
- **Response** – A `response` element describes the output that results from performing an HTTP method on a resource.
- **Parameter** - A `param` element describes a parameterized component of its parent element. A `param` element can either be a parameter definition or a reference to a parameter defined elsewhere.
- **Option** - An `option` element defines one of a set of possible values for the parameter represented by its parent `param` element.

4 MGH Center for the Development of a Virtual Tumor (CViT) Overview

Digital Model Repository (DMR)

- ***RDF-based Semantic database to store models and model metadata***
- ***RDF links to other web-based resources produces a semantic graph***

CViT.org Web Application

- ***Provides the graphical user interface to the repository***
- ***eLicensing Workflow protects intellectual property***
- ***Scientists can add new models, share models, and discuss model simulations***
- ***Allows for execution of the models in the repository***

caBIG DMR Data Service

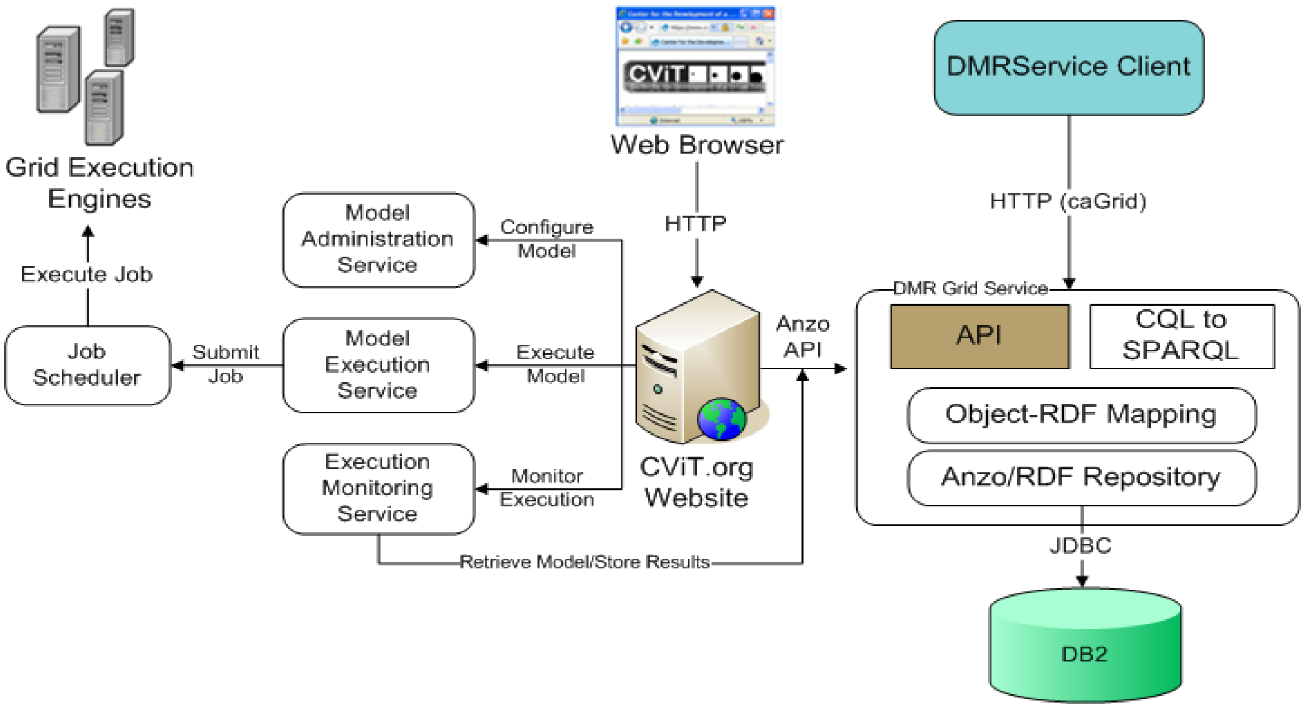
- ***Silver-Level Compliant caBIG Data Service***
- ***Securely access DMR through caGrid***
- ***eLicensing Workflow preserved through caGrid***

Computational Model Execution Framework

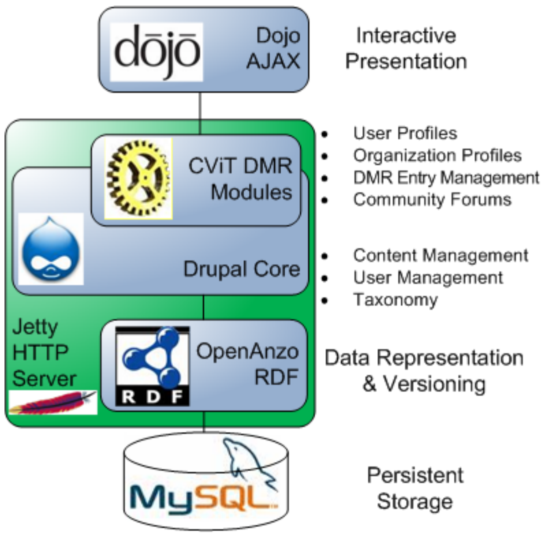
- ***Models annotated with execution metadata***
- ***Grid-based execution of the models in the repository***

MGH-CViT DMR and CMEF Combined Architecture

The MGH-CViT DMR and CMEF together can either run on a single computer, or on the production servers at MGH/CViT, the DMR (plus caBIG Data Service) is installed on one computer, the CMEF is installed on a second computer, and additional Condor grid execution nodes are installed to handle model execution. The CMEF computer can be configured to execute models (as a grid execution node) or forward jobs to separate nodes.



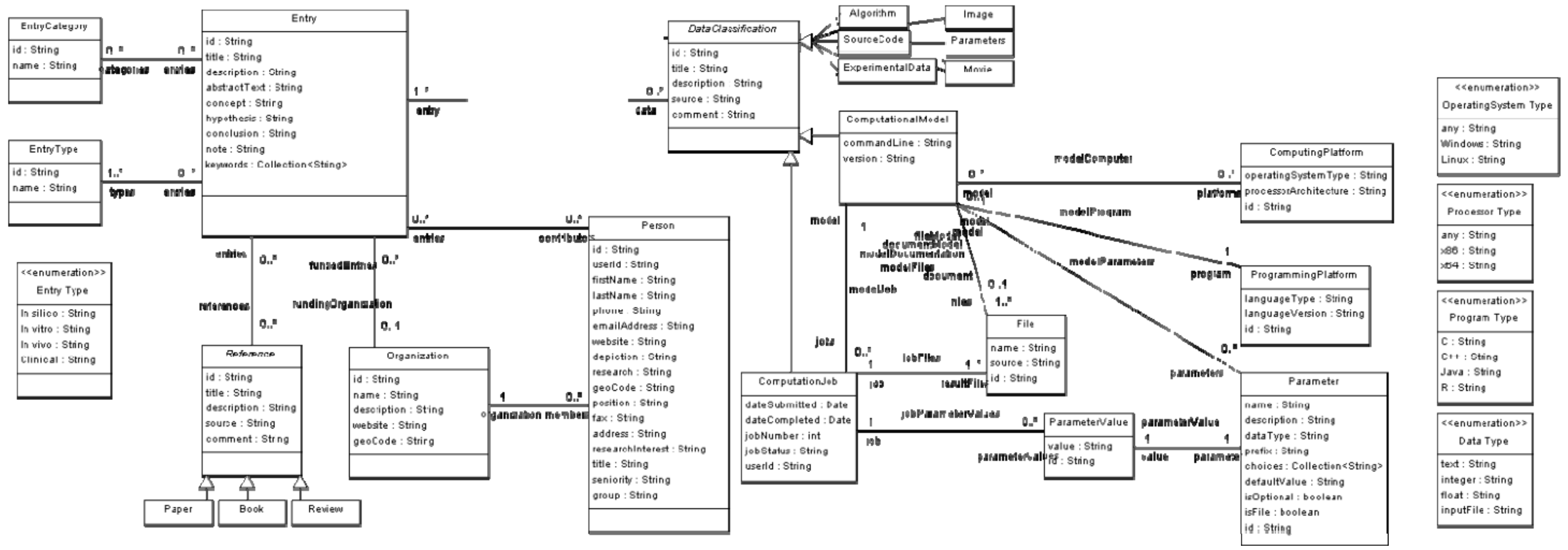
The MGH-CViT DMR uses Resource Description Framework (RDF) as the foundational means to model and link entities corresponding to real-world concepts, for the DMR, it uses RDF to represent computational models of cancer and links these models to publications, experiments, and data built into a provenance structure. RDF is a standard model for data interchange on the Web. RDF facilitates data merging even when the underlying schemas differ and supports the evolution of schemas over time without requiring changes to the persistent storage and APIs accessing the data. In general, RDF is more robust against schema evolution than relational and XML approaches In addition, RDF allows structured and semi-structured data to be mixed, exposed, and shared across different applications and linked with external websites. RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link (referred to as a "triple").



Component	Description
Dojo	The Dojo Toolkit is an open source modular JavaScript library designed to ease the rapid development of cross platform, JavaScript/Ajax based applications and web sites. Dojo is available under either the terms of the modified BSD license or the Academic Free License version 2.1. (Dojo Foundation 2009). **The CViT DMR utilizes Dojo to facilitate interactive user interfaces and data transfer.**
Drupal	Drupal is a free software package that allows a community of users to easily publish, manage and organize a wide variety of content on a website. Drupal is open-source software distributed under the GPL ("GNU General Public License") and is maintained and developed by a community of thousands of users and developers (Drupal 2009). **CViT's DMR utilizes custom Drupal modules to support collaboration and the CViT community.**
Anzo	**Anzo is an open source enterprise-featured RDF store and service oriented middleware platform that provides support for multiple users, distributed clients, offline work, real-time notification, named-graph modularization, versioning, access controls, and transactions with preconditions. At the center of an Anzo system is a server capable of storing many millions of RDF triples in an underlying relational database (OpenAnzo 2009). CViT's DMR was one of the first productions systems to utilize Anzo for RDF storage.**
Apache HTTPD	The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows. The goal of this project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards. Apache has been the most popular web server on the Internet since April 1996 (Apache Software Foundation 2009). **The CViT Digital Model Repository employs the Apache HTTPD server along with Drupal and Dojo to manage the Web-based interface to CViT.**
MySQL	MySQL is a relational database management system (RDBMS) which has more than 11 million installations. The project's source code is available under terms of the GNU General Public License, as well as under a variety of proprietary agreements (MySQL 2009). **While IBM DB2 is the primary production database for the CViT Digital Model Repository, MySQL has been successfully used in development and test installations of the DMR.**

MGH-CViT DMR and CMEF (Combined Domain Model)

The conceptual data model of DMR and CMEF is shown in the diagram below. In the following tables we provide details about the entities and attributes of this model.



Entry	
Description	An entry in the repository contains relevant, uploaded information regarding a project. Entries can only be created by Principal Investigators (PIs). With a Licensing Officer's (LO's) approval a PI can have his entry published to other users of the repository. Designated contributors can annotate entries with metadata.
NCI Concept Code	C47885
id	
Description	A unique entry identifier
Data Type	String
NCI Concept Code	C25364
title	
Description	A name to identify the entry
Data Type	String
NCI Concept Code	C42774
description	
Description	A paragraph explaining why the entry exists and what project it contains. This defines the scope for further data uploaded to the entry.
Data Type	String
NCI Concept Code	C25365
abstractText	
Description	A brief summary of the project's description
Data Type	String
NCI Concept Code	C60765
concept	
Description	Background and basic idea of this project
Data Type	String
NCI Concept Code	C48910
hypothesis	
Description	What assumption(s) will be proved by this experiment

Data Type	String
NCI Concept Code	C28362
conclusion	
Description	The outcome and significance of the project
Data Type	String
NCI Concept Code	C54033
note	
Description	Space for adding notes
Data Type	String
NCI Concept Code	C42619
keywords	
Description	List of 3-5 searchable terms characterizing the entry
Data Type	Collection<String>
NCI Concept Code	C43513
Associations	
fundingOrganization	
Description	A source of funding for this particular entry
Data Type	Organization
NCI Concept Code	C19711, C17769
contributors	
Description	Repository users with write access to this entry
Data Type	Collection<Person>
NCI Concept Code	C25190
references	
Description	Upload, link to, or select (link within repository) references
Data Type	Collection<Reference>
NCI Concept Code	C25641

data	
Description	Data files can be included in an entry by way of upload, http reference, or internal repository reference
Data Type	Collection<DataClassification>
NCI Concept Code	C25474
types	
Description	Model entry classification denotes the origin of the data
Data Type	Collection<EntryType>
Value Domain	Computation | Invitro | Invivo | Clinical
NCI Concept Code	C25284, C25474, C47885
categories	
Description	A collection of values from the Entry Category enumeration describing the Entry.
Data Type	Collection<EntryClassification>
NCI Concept Code	C25372, C47885

Person	
Description	User profile
NCI Concept Code	C25190
id	
Description	A unique user identifier
Data Type	String
NCI Concept Code	C25364
userId	
Description	The persons DMR user id
Data Type	String
NCI Concept Code	C42694
title	

Description	Suffix titles. Usually Ph.D. or M.D.
Data Type	String
Value Domain	Ph.D. | M.D. | Sc.D.
NCI Concept Code	C25354
firstName	
Description	User's first name
Data Type	String
NCI Concept Code	C40974
lastName	
Description	User's last name
Data Type	String
NCI Concept Code	C40975
phone	
Description	Phone number
Data Type	String
NCI Concept Code	C40978
emailAddress	
Description	Email address
Data Type	String
NCI Concept Code	C42775
website	
Description	User's homepage (URL)
Data Type	String
NCI Concept Code	C19467
depiction	
Description	User's icon/picture (URL)
Data Type	String

NCI Concept Code	C54273
research	
Description	Type of research being done by user. (used in cvit.org/mashup)
Data Type	String
Value domain	experimental | computational | both | nci
NCI Concept Code	C25284, C15429
geoCode	
Description	Coordinates for placing organizations on cvit.org/mashup
Data Type	String
NCI Concept Code	C25341, C68643, C68642
position	
Description	Job title
Data Type	String
NCI Concept Code	C19067, C25193
fax	
Description	Fax number
Data Type	String
NCI Concept Code	C42879
address	
Description	Current mailing address
Data Type	String
NCI Concept Code	C70946
researchInterest	
Description	Several paragraphs describing current research interests
Data Type	String

NCI Concept Code	C48910, C15429
group	
Description	Tracks groups of cvit users for cvit.org/teampages
Data Type	String
Value Domain	Main | AdvisoryBoard | NCI | ICBP | Unlisted
NCI Concept Code	C41167
seniority	
Description	Tracks a user's expertise level
Data Type	String
Value Domain	Faculty | Postdoc | GradStudent | Other
NCI Concept Code	C25554, C25193
Associations	
entries	
Description	List of Entries for which this Person is listed as a contributor. See: Entry.contributors
Data Type	Collection<Entry>
NCI Concept Code	C47885
organization	
Description	Institution with which user is affiliated. Within the DMR, users can only belong to one organization.
Data Type	Organization
NCI Concept Code	C19711

EntryCategory	
Description	Enumerates a set of classifications to facilitate discovery and retrieval of entries. Each entry may be tagged with an Entry Category so that it can be found in the classification tree.
NCI Concept Code	C47885, C25372
id	

Description	A unique data classification identifier
Data Type	String
NCI Concept Code	C25364
name	
Description	The name for the category
Data Type	String
NCI Concept Code	C42614
Associations	
entries	
Description	List of Entries for which this EntryCategory is relevant
Data Type	Collection<Entry>
NCI Concept Code	C47885

EntryType	
Description	Model entry classification denotes the origin of the data. (Computation } Invitro | Invivo | Clinical)
NCI Concept Code	C47885, C25474, C25284
id	
Description	A unique data classification identifier
Data Type	String
NCI Concept Code	C25364
name	
Description	The name of the Entry Type. One of In silico, In vitro, In vivo, Clinical
Data Type	String
NCI Concept Code	C42614
Associations	
entries	
Description	List of Entries for which this EntryType is relevant.

Data Type	Collection<Entry>
NCI Concept Code	C47885

DataClassification	
Description	Data files can be included in an entry by way of upload, http reference, or internal repository reference.
NCI Concept Code	C25474,C25161
id	
Description	A unique data classification identifier
Data Type	String
NCI Concept Code	C25364
title	
Description	Name of the upload
Data Type	String
NCI Concept Code	C42774
description	
Description	A brief description of the file's contents
Data Type	String
NCI Concept Code	C25365
source	
Description	Link to the file itself (URL)
Data Type	String
NCI Concept Code	C42743
comment	
Description	Any additional user input
Data Type	String
NCI Concept Code	C25393

Associations	
entry	
Description	List of Entries for which this Data is relevant. See: Entry.data
Data Type	Collection<Entry>
NCI Concept Code	C47885

Algorithm: DataClassification	
Description	Tag indicates uploaded data is an algorithm
NCI Concept Code	C16275

SourceCode: DataClassification	
Description	Tag indicates uploaded data is source code
NCI Concept Code	C47901

Parameters: DataClassification	
Description	Tag indicates uploaded data is a set of parameters
NCI Concept Code	C48913

Image: DataClassification	
Description	Tag indicates uploaded data is an image
NCI Concept Code	C48179

Movie: DataClassification	
Description	Tag indicates uploaded data is a movie (e.g., simulation, microscopy)
NCI Concept Code	ObjectClassConceptCode: C75001

ExperimentalData: DataClassification	
Description	Tag indicates uploaded data is experimental data
NCI Concept Code	ObjectClassConceptCode: C25474 ObjectClassQualifierConceptCode1: C42790

Reference	
Description	Link to a bibliographical reference. This can point to a PDF for upload, a PubMed id, or a reference already in the repository.
NCI Concept Code	C25641
id	
Description	A unique reference identifier
Data Type	String
NCI Concept Code	C25364
title	
Description	Name of reference
Data Type	String
NCI Concept Code	C42774
description	
Description	Details about/in the reference
Data Type	String
NCI Concept Code	C25365
source	
Description	Link to the reference file (URL)
Data Type	String
NCI Concept Code	C42743
comment	
Description	Additional comments from user
Data Type	String
NCI Concept Code	C25393
Associations	
entries	

Description	List of Entries for which this Reference is relevant. See: Entry.reference
Data Type	Collection<Entry>
NCI Concept Code	C47885

Book: Reference	
Description	Tag indicates uploaded reference is a book
NCI Concept Code	C16360

Paper: Reference	
Description	Tag indicates uploaded reference is a paper
NCI Concept Code	C47902

Review: Reference	
Description	Tag indicates uploaded reference is a review
NCI Concept Code	C47902,C42729

Organization	
Description	An institution. Most of these will be colleges, universities, and research institutes. Each user in the repository is affiliated with one and only one organization. Each organization has one or more licensing officers to approve of user's licensing requests.
NCI Concept Code	C19711
id	
Description	A unique organization identifier
Data Type	String
NCI Concept Code	C25364
name	
Description	Name of the organization
Data Type	String
NCI Concept Code	C42614

description	
Description	Brief description of the organization
Data Type	String
NCI Concept Code	C25365
website	
Description	Organization's homepage (URL)
Data Type	String
NCI Concept Code	C19467
geoCode	
Description	Geographical location of organization in “lat, long” format. Used in cvit.org/mashup
Data Type	String
NCI Concept Code	C25341, C68643, C68642
Associations	
members	
Description	People associated with the Organization. Within the DMR, users can only belong to one Organization. See: Person.organization
Data Type	Collection<Person>
NCI Concept Code	C25190
fundedEntries	
Description	An entry funded by a particular organization
Data Type	Collection<Entry>
NCI Concept Code	C47885,C17769

ComputationalModel: DataClassification	
Description	Extends from DataClassification to encapsulate the content and metadata of an executable computational model
name	
Description	The name of the model

Data Type	String
description	
Description	Describes the operation of the model
Data Type	String
commandLine	
Description	Command line used to execute the model. Values enclosed in angle brackets are replaced by corresponding ParameterValues before the model is executed
Data Type	String
version	
Description	Software version of the computational model
Data Type	String
Associations	
modelFiles	
Description	Source files that constitute the computational model and requisite executable files. For example, a Java archive or a program configuration file
Data Type	Collection<File>
modelDocumentation	
Description	User's Guide or documentation describing the use of the computational model
Data Type	File
modelComputer	
Description	Operating system and CPU constraints required by the model
Data Type	ComputingPlatform
modelProgram	
Description	Program execution language constraints (Java, Perl, R, C++, etc.) required by the model
Data Type	ProgrammingPlatform
modelParameter	
Description	Program parameters that can be set for the model
Data Type	Collection<Parameter>

File	
Description	References a file
Name	
Description	The file’s name
Data Type	String
Source	
Description	The file’s location
Data Type	URL

ComputingPlatform	
Description	Describes model program execution hardware constraints
operatingSystemType	
Description	Indicates the operating system required to run the model (Any | Windows | Linux)
Data Type	OperatingSystem Type
processorArchitecture	
Description	Indicates the required CPU type to run the model(Any | x86 | x64)
Data Type	Processor Type

ProgrammingPlatform	
Description	Describes model program execution environment constraints
languageType	
Description	Classifies the programming language of the model (C | C++ | Java | R)
Data Type	Program Type
languageVersion	
Description	The minimum version of the programming language supported by the model
Data Type	String

Parameter

Description	Defines the metadata for describing an individual input value for a computational model. Parameters may be input files or values entered on the command line. See ComputationalModel::commandLine
Name	
Description	The name of the parameter – for a file, it should be the file's name, for a command line parameter, it should match the name specified in commandLine, for example "<*name*>"
Data Type	String
Description	
Description	Description of the parameter or value constraints
Data Type	String
dataType	
Description	Defines the required parameter value type (Text | Integer | Float | File)
Data Type	Data Type
Prefix	
Description	A command line prefix that will be added if the value is present. For example "-F " or "-o "
Data Type	String
Choices	
Description	Set of values that constrain the input parameter
Data Type	Collection<String>
defaultValue	
Description	Default value used during model execution if no value is specified
Data Type	String
isOptional	
Description	Indicates that the parameter value can be omitted (if true)
Data Type	Boolean
isFile	
Description	Indicates that the parameter value is a file
Data Type	Boolean

ComputationJob: DataClassification	
Description	Extends from DataClassification to encapsulate the content and metadata of an executing (or executed) computational model
dateSubmitted	
Description	The date/time that the computation job was submitted for execution
Data Type	Date
dateCompleted	
Description	The date/time that the computation job execution completed
Data Type	Date
jobNumber	
Description	System-assigned number identifying the model execution job
Data Type	Integer
jobStatus	
Description	Result of running the model (Success | Failure)
Data Type	String
userId	
Description	Identifier of the user who initiated the ComputationJob
Data Type	String
Associations	
jobParameterValue	
Description	Values set for each Parameter of the model for the job
Data Type	Collection<ParameterValue>
jobFiles	
Description	Files produced by executing the model computation job, may include execution log, console output, and output files
Data Type	Collection<File>

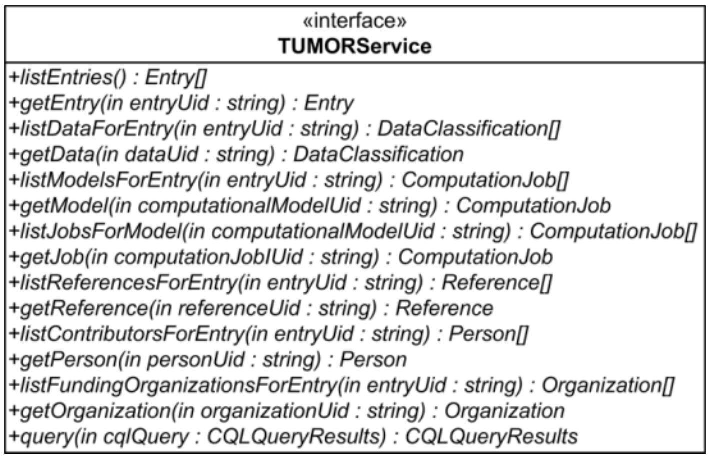
ParameterValue

Description	Value that will be utilized during the computation job for the given parameter
Value	
Description	Value that will be utilized during the computation job for the given parameter. Input files should be specified by URLs
Data Type	String
Associations	
parameterValue	
Description	Parameter that this value sets
Data Type	Parameter

MGH-CViT DMR TUMOR Web Service

PROVIDES READ-ONLY ACCESS TO AUTHORIZED MODELS IN DMR.

Based on the planned clinical scenarios (D2.1), the MGH-CViT Digital Model Repository must be extended to provide the required functionality to provide the models and model metadata stored in the CViT repository to the EU/TUMOR platform for remote execution. The TUMOR clinical scenario indicates a read-only interface to the models, metadata, and simulation results are sufficient to support the requirements. The TUMOR clinical scenarios do not envision altering the content of the CViT repository.



```
«interface»
TUMORService
+listEntries() : Entry[]
+getEntry(in entryUid : string) : Entry
+listDataForEntry(in entryUid : string) : DataClassification[]
+getData(in dataUid : string) : DataClassification
+listModelsForEntry(in entryUid : string) : ComputationJob[]
+getModel(in computationalModelUid : string) : ComputationJob
+listJobsForModel(in computationalModelUid : string) : ComputationJob[]
+getJob(in computationJobIUid : string) : ComputationJob
+listReferencesForEntry(in entryUid : string) : Reference[]
+getReference(in referenceUid : string) : Reference
+listContributorsForEntry(in entryUid : string) : Person[]
+getPerson(in personUid : string) : Person
+listFundingOrganizationsForEntry(in entryUid : string) : Organization[]
+getOrganization(in organizationUid : string) : Organization
+query(in cqlQuery : CQLQueryResults) : CQLQueryResults
```

TUMORService	
Description	The MGH-CViT Digital Model Repository provides authenticated access to computational models, model metadata, and model simulation results through a TUMOR-specific Web Service API. The TUMOR Web Service API extends the DMR caBIG Data Service API by providing additional methods to retrieve computational models and computation job results executed by the Computational Model Execution Framework.
Note:	All methods in the TUMORService API can be reproduced through the “`CQLQueryResults query(CQLQuery cqlQuery)`” method. The additional methods are provided for convenience.

listEntries
Enumerates the set of Entries in the DMR that the user has authorization to read.
Parameters
none

Returns
Entry[] – list of all Entries in the repository, filtered by user authorization
Exceptions
LoginException – if authenticated session has not been established.
AuthorizationException – if User does not have sufficient privilege to perform this action.

getEntry
Returns the Entry specified by the provided Entry UID.
Parameters
String entryUid – unique Entry identifier
Returns
Entry – specific Entry identified by entryUid, null if Entry does not exist
Exceptions
LoginException – if authenticated session has not been established.
AuthorizationException – if User does not have sufficient privilege to perform this action.

listDataForEntry
Enumerates the set of DataClassification associated with the given Entry UID that the user has authorization to read.
Parameters
String entryUid – unique Entry identifier
Returns
DataClassifcation[] – array of DataClassification sub-classes that are associated with the specified Entry, filtered by user authorization
Exceptions
LoginException – if authenticated session has not been established.
AuthorizationException – if User does not have sufficient privilege to perform this action.

getData

Returns the DataClassification specified by the provided DataClassification UID
Parameters
String dataClassificationUid – unique DataClassification identifier
Returns
DataClassification – DataClassification subclass identified by dataClassificationUid
Exceptions
LoginException – if authenticated session has not been established.
AuthorizationException – if User does not have sufficient privilege to perform this action.

listModelsForEntry
Enumerates the set of ComputationalModel associated with the given Entry UID that the user has authorization to read.
Parameters
String entryUid – unique Entry identifier
Returns
ComputationalModel[] – array of ComputationalModel classes that are associated with the specified Entry, filtered by user authorization
Exceptions
LoginException – if authenticated session has not been established.
AuthorizationException – if User does not have sufficient privilege to perform this action.

getModel
Returns the ComputationalModel specified by the provided ComputationalModel UID.
Parameters
String computationalModelUid – unique ComputationalModel identifier
Returns
ComputationalModel – ComputationalModel identified by computationalModelUid
Exceptions
LoginException – if authenticated session has not been established.

AuthorizationException – if User does not have sufficient privilege to perform this action.

listJobsForModel
Enumerates the set of ComputationJob associated with the given ComputationalModel UID that the user has authorization to read.
Parameters
String computationalModelUid – unique ComputationalModel identifier
Returns
ComputationJob[] – array of ComputationJob classes associated with the specified ComputationalModel, filtered by user authorization
Exceptions
LoginException – if authenticated session has not been established.
AuthorizationException – if User does not have sufficient privilege to perform this action.

getJob
Returns the ComputationJob specified by the provided ComputationJob UID
Parameters
String computationJobUid – unique ComputationJob identifier
Returns
ComputationJob – ComputationJob identified by computationJobUid
Exceptions
LoginException – if authenticated session has not been established.
AuthorizationException – if User does not have sufficient privilege to perform this action.

listReferencesForEntry
Enumerates the set of Reference associated with the given Entry UID that the user has authorization to read.
Parameters
String entryUid – unique Entry identifier
Returns

Reference[] – array of Reference sub-classes that are associated with the specified Entry, filtered by user authorization
Exceptions
LoginException – if authenticated session has not been established.
AuthorizationException – if User does not have sufficient privilege to perform this action.

getReference
Returns the Reference specified by the provided Reference UID
Parameters
String referenceUid – unique Reference identifier
Returns
Reference – Reference sub-class identified by referenceUid
Exceptions
LoginException – if authenticated session has not been established.
AuthorizationException – if User does not have sufficient privilege to perform this action.

listFundingOrganizationsForEntry
Enumerates the set of Organizations associated with the given Entry UID under the role ‘funding organization’
Parameters
String entryUid – uniqe Entry identifier
Returns
Organization[] – array of Organization classes associated with the specified Entry as a Funding Organization, filtered by user authorization
Exceptions
LoginException – if authenticated session has not been established.
AuthorizationException – if User does not have sufficient privilege to perform this action.

getOrganization
Returns the Organization specified by the provided Organization UID

Parameters
String organizationUid – unique Organization identifier.
Returns
Organization – Organization identified by organizationUid, null if Organization does not exist.
Exceptions
LoginException – if authenticated session has not been established.
AuthorizationException – if User does not have sufficient privilege to perform this action.

listContributorsForEntry
Enumerates the set of Person associated with the given Entry UID under the role 'contributor'
Parameters
String entryUid – unique Entry identifier
Returns
Person[] – array of Person classes associated with the Entry as a Contributor, filtered by user authorization.
Exceptions
LoginException – if authenticated session has not been established.
AuthorizationException – if User does not have sufficient privilege to perform this action.

getPerson
Returns the Person specified by the provided Person UID
Parameters
String personUid – unique Person identifier
Returns
Person – Person class specified by persionUid, null if Person does not exist.
Exceptions
LoginException – if authenticated session has not been established.
AuthorizationException – if User does not have sufficient privilege to perform this action.

query
Executes a caBIG Query Language (CQL) query against the information in the repository. Only records that the User has read access to will be returned in the result set.
Parameters
CQLQuery cqlQuery – the CQL query to execute against the repository.
Returns
CQLResult – the result of executing the query against the repository.
Exceptions
LoginException – if authenticated session has not been established.
AuthorizationException – if User does not have sufficient privilege to perform this action.
IllegalArgumentException – if CQLQuery is improperly formatted.
See Also
DMR Use Case #3.8