metafor Common Metadata for Climate Modelling Digital Repositories

METAFOR CIM Web Service Implementation
METAFOR Deliverable 5.5 M27

PROJECT	
Project acronym	METAFOR
Project full title	Common Metadata for Climate Modelling Digital Repositories
Grant agreement no:	211753
Funding Scheme	Combination of Collaborative Projects & Coordination and Support Actions
Call Topic	INFRA-2007-1.2.1 Scientific Digital Repositories
DOCUMENT	
Deliverable	D5.5 Month 27
Deliverable Title	Web Service Implementation
Document Identifier	METAFOR-D5.5_M27
Date	August 30th 2010
Work Package	WP4 CIM Web Services
Authors	IPSL
Document Status	Draft
Document Link	http://metaforclimate.eu/documents/WP4/D5.5.odt

Nature 1		
R	Report	X
P	Prototype	
D	Demonstrator	
O	Other	

Document History			
Version	Date	Comment	Author/Partner
0.1	Aug 02nd 2010	Draft	M. Morgan/IPSL
0.2	Aug 06th 2010	Draft	M. Morgan/IPSL
0.3	Aug 13th 2010	Draft	M. Morgan/IPSL, A. Treshansky/BADC
1.0	Aug 25th 2010	Review	M. Morgan/IPSL, A. Treshansky/BADC

CAPACITIES

e-infrastructure

Table of Contents

1. SUMMARY 3
2. ENVIRONMENT 4
3. MESSAGING WORKFLOW 5
4. SOFTWARE ARCHITECTURE 6
   - Apache Filters 6
   - REST Controllers 6
   - Components 6
   - Resource Managers 6
5. CROSS CUTTING ASPECTS 7
   - Hosting 7
   - Execution 7
   - Security 7
   - REST & AtomPub 7
   - Base URI 7
   - Metrics 7
   - Instrumentation 7
6. APACHE FILTERS 8
   - WSGI 8
   - ESG-F Security 8
7. REST CONTROLLERS 9
   - Document 9
   - Query 10
   - Validation Report 12
   - Controlled Vocabulary 13
   - Tracking 13
8. COMPONENTS 15
   - Security 15
   - Data Access 15
   - Validation 15
   - Models 15
   - Utilities 15
9. RESOURCE MANAGERS 16
   - eXist dB 16
   - PostGres dB 16
   - SMTP 16
   - File System 16
10. BIBLIOGRAPHY (LINKS VALID AS OF 01ST AUGUST 2010) 17

1. Summary

The Metafor web services have been implemented according to the Service Orientated Architecture (SOA) paradigm. SOA is a key tenet in the design of secure, robust & distributed systems. Such an architecture deconstructs a system into a set of discrete functional units known as services. Services are said to be supplied by providers (e.g. Metafor) and consumed by clients (i.e. tools, applications, and in some cases other services).

A web service is such a discrete functional unit deployed upon a web server and thus consumable via the HTTP protocol. Web services leverage inherent HTTP features such as security & caching. By decomposing a system into web-services, rich & diverse informational eco-systems can be incubated, nurtured & supported.

Web services can be implemented in several different styles, but an architectural style known as REST (REpresentational State Transfer) is particularity well suited to the HTTP protocol. REST services posit resources as the unit of exchange between a service & client. A resource is an informational unit supporting at least one representation (i.e. an encoding such as XML)

In order to deconstruct a system into a set of discrete REST services, one identifies the collection of resources flowing through the system. Each type of resource is then managed by a distinct REST service implementation.

The Metafor Common Information Model, i.e. the CIM, is an ontology designed to become the ipso-facto standard for climate modelling related metadata. The CIM eco-system allows institutes to integrate the CIM into their day to day climate modelling processes. It achieves this by supporting various requirements: the ontology itself; validation; search; dissemination; integration (with other metadata platforms such as Earth System Grid).

These requirements are met via the implementation of Metafor web services. These web services are RESTful in style and support a diverse array of CIM resources: documents; validation reports; queries, tracking id's, controlled vocabularies.

Collectively CIM web services constitute a platform upon which CIM clients such as web-portals or a standalone tools can be developed. At the heart of a CIM client is the consumption of one or more CIM web services, at the heart of a CIM web-service is a CIM resource.

This document details the Metafor web services implementation by outlining the following:

1. An overview of the environment in which the Metafor web services operate;
2. The typical service/client messaging workflow;
3. A high-level view of the employed software architecture.
4. Cross cutting aspects, such as security, common to all Metafor web services;
5. A low-level view of each layer within the software architecture;
6. A bibliography of links to various web-sites and pages of interest.

2. Environment

Institutes generate vast amounts of *output data* as a result of running simulations against climate models in support of experiments. This output data typically resides at a number of locations managed by the institute, these locations are known as *data-nodes*. Output data is of two types, i.e. *core* (available to the public domain) & *operational* (available only to members of the institute).

Output data is so voluminous, i.e. petabyte scale, that moving it across networks is both slow & error prone. This causes serious bottlenecks in the e-science process. Resolving this bottleneck is a key objective for institutes wishing to support use cases such as search, dissemination & collaboration. Supporting such use cases are key to effective e-science strategies.

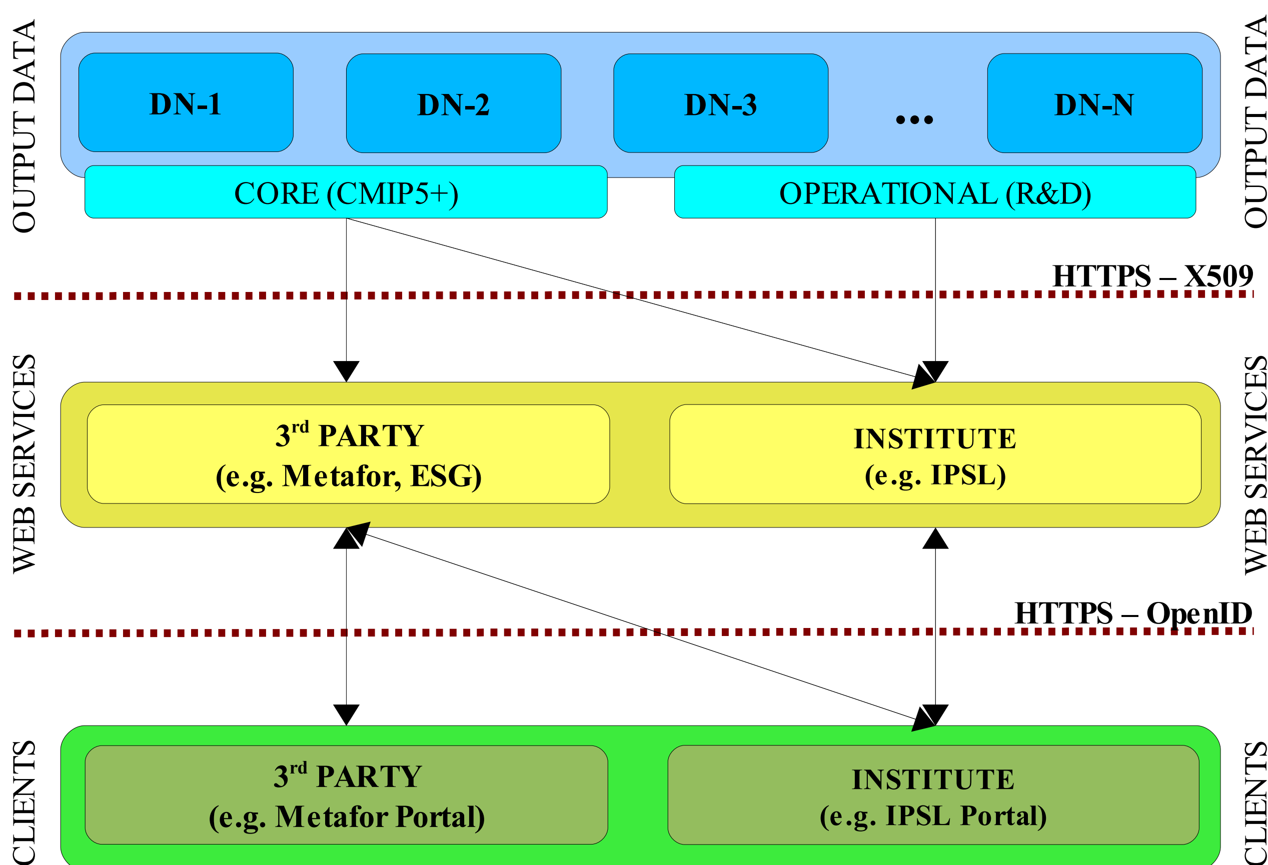
Metadata is the solution to resolving this bottleneck. Such metadata, generated from the output data and published to web services that manage digital repositories, includes not only descriptions of climate data, but also of their provenance - the models used to generate data, the simulations developed using those models, the experiments for which the simulations were run, etc.

Standalone tools typically perform the work of metadata generation & publication, they are in fact specialized web service clients. The communication channel between such a standalone tool running on a data-node & a web service is secured via HTTPS encryption and X509 certificates.

Metadata generated from core output data is published to web services run by 3rd party groups such as Metafor. Metadata generated from operational output data is published to web services run by the institute. Web services typically store such published metadata in databases.

Such web services typically also support search, this is leveraged by clients such as web portals, that may be developed either by the institute or by 3rd parties (e.g. Metafor). In this scenario communication security is via HTTPS encryption and OpenID tokens.

At least *25 institutes world-wide* are participating in the CMIP5 process. Thus the environment in which Metafor CIM web services operate is complex and involves many actors. In order to successfully operate within this environment, CIM web services must support the development of tools & applications that add real value to the climate science community.



3. Messaging Workflow

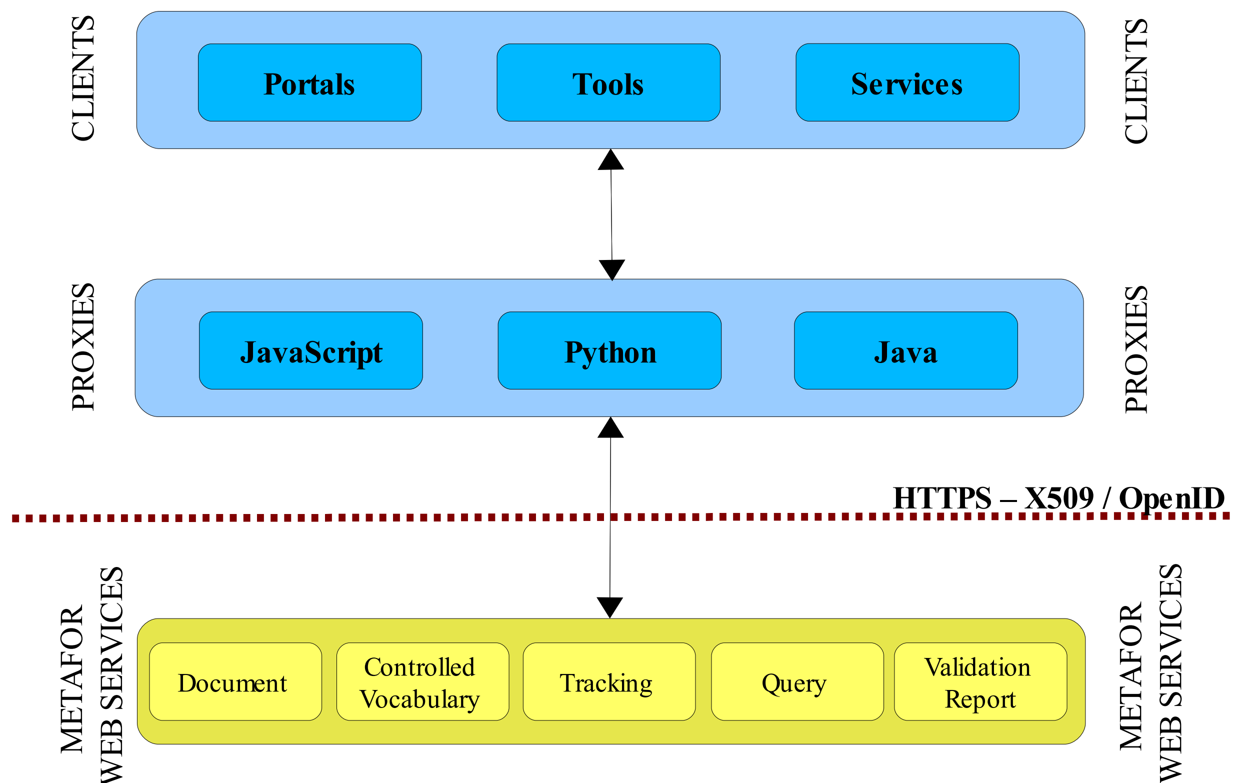
Message exchange is at the core of the interaction between a client and a service. Clients send request messages over a secure communication channel to services that reply with response messages, i.e. the classic request/response message pattern.

In order to send a request to a service, the client typically prepares a request, opens a communication connection, dispatches the request, and awaits a response. It also deals with error scenarios when the communication process fails, e.g. timeouts, network failures ...etc.

A specialized software component called a proxy takes care of low level messaging details such as connection, dispatch, error handling...etc. Clients are developed in an array of programming languages, thus proxies must reflect this diversity and also be implemented in an array of programming languages (e.g. Javascript, Java, Python ...etc.). Metafor provides such proxy libraries in order to smooth adoption of its services.

Within the context of web services, the communication channel between the client & service is based on the HTTP protocol. The HTTP protocol is the backbone of the internet and provides many in-built features such as encryption, authentication, caching, compression ...etc. It is secured at the transport level via encrpytion, see HTTPS. It is secured at the client/service interaction level via authentication techniques, e.g. X509 certificates, OpenID tokens...etc.

Upon receipt of an HTTP request the web service will perform a unit of work and return a response to the client. Error scenarios typically occur in one of two situations: the request is invalid; an unexpected processing error occurs. In either error scenario the client is informed and thus can take appropriate compensatory action.



Client Side: Clients send an HTTP request over an HTTPS channel via a proxy to a CIM web service. Clients authenticate by attaching either an X509 certificate or OpenID token to the request.

Service Side: The received request is routed to a web server for processing by the web service. The web service performs the relevant unit of work and returns an HTTP response to the client.

4. Software Architecture

The CIM web service software architecture is based upon a multi-layer architecture. Each layer in such an architecture has a limited set of responsibilities and collaborates with the layer directly below. Each layer has no knowledge of the layer above it and no understanding of how the layer below it does it's work. Such an architecture promotes both flexibility & reusability as well as providing defence in depth against attacks. There are 4 layers within the CIM web services software stack:

Apache Filters

The Apache web server application supports the request filtering. CIM web services leverage two filters: WSGI filter for passing requests to the REST controllers in the layer below; ESG Federation security filter.

REST Controllers

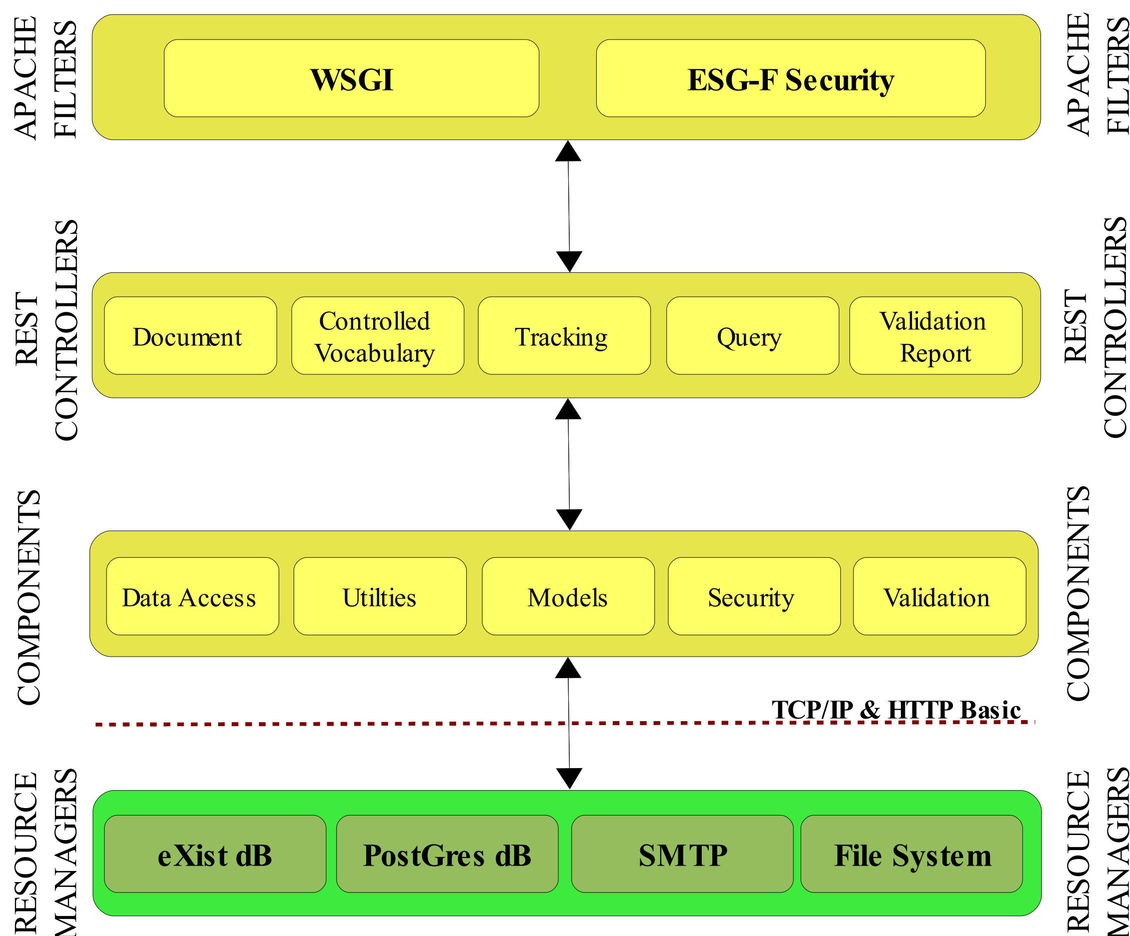
CIM web services are implemented as REST controllers using the Pylons web framework. Such controllers are implemented using the Python programming language and typically invoke the component layer below.

Components

Python based components perform most of the low level request processing. This involve invoking resource managers in order to interact with resources such as databases.

Resource Managers

A wide array of resource managers permit the CIM web services to save & retrieve information in a variety of formats for a variety of purposes.



5. Cross Cutting Aspects

Hosting

CIM web services are hosted within a web farm in which each machine runs the Linux operating system & the *Apache* web server application. Incoming HTTP requests are routed to a web server where they are intercepted and filtered by Apache.

Execution

HTTP requests, filtered by Apache, are passed to *controllers* for execution. Controllers orchestrate calls to *components* in order to perform units of work (e.g. a database lookup).

Security

There are 4 levels of web-service security, all must be complied with:

1. All requests / responses must be *encrypted*;
2. All clients must be *authenticated* to use a service;
3. All clients must be *authorized* to invoke a service operation;
4. All requests must be *validated* in order to prevent attacks by malicious users.

CIM web-services achieve each of these levels thus:

1. Via the *HTTPS* communications protocol;
2. Via an Apache filter supporting authentication with *OpenId tokens* or *X509 certificates*.
3. Via an Apache filter supporting authorization with *SAML* (Security Assertion Markup Language).
4. Via a security component that validates all request headers & content.

Note - the Apache filter performing levels 2 & 3 is developed by the Earth System Grid Federation.

REST & AtomPub

At the core of REST-ful services are *resources*, an array of CIM related resource types are supported: documents; validation reports; queries...etc. *AtomPub* is an XML messaging protocol based on RESTful principles. Whilst AtomPub messages must adhere to the AtomPub XML standard, the inner content of an AtomPub message may be in any format (e.g. XML, JSON, BINARY). CIM documents lend themselves well to being exposed as AtomPub services. Many Metafor web services will take as input an AtomPub message.

Base URI

All CIM web services share a common *base URI* : www.metaforcliemate.eu/services/ All URI's referred to in this document are assumed to be prefixed by this base URI unless otherwise stated.

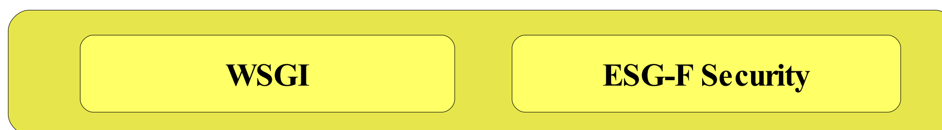
Metrics

Gathering metrics upon CIM web-services usage is achieved by intercepting the execution of each CIM web service operation. The interceptor passes information to a specialized metrics component that saves the metrics information to a database & periodically generates metrics reports.

Instrumentation

To monitor system health an instrumentation sub-system is implemented via a system event logging mechanism. High level events, generic in content, are logged via Apache web server. Low level events, detailed in content, are logged via in-built Python functions. Event logs are available for secure viewing.

6. Apache Filters



Apache filters can either filter incoming HTTP requests (input filters) or outgoing HTTP responses (output filters). Apache extensively uses filters internally in order to perform operations such as compression, encryption, caching, etc. Apache is installed upon all Metafor web servers and two Apache input filters are installed in order to perform generic HTTP request processing.

WSGI

This filter supports the hosting of WSGI (Web Server Gateway Interface) compatible applications and is suitable for use in hosting high-performance websites. Pylons is one such WSGI compatible application and is the Python based framework upon which all Metafor web services are built.

The Apache WSGI filter routes incoming HTTP requests to the Metafor web services Pylons application for further processing. Routed requests are processed by so-called Pylons controllers which perform a unit of work and return an HTTP response.

ESG-F Security

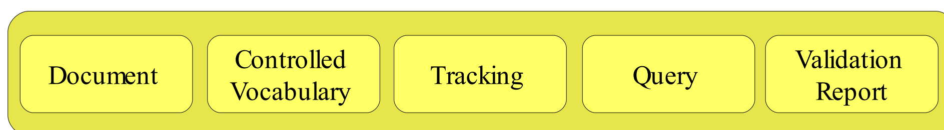
The Earth System Grid Federation (ESG-F) provides a federated security infrastructure for all applications and services that participate in the IPCC CMIP5 process. One key objective of this infrastructure is to support so-called single-sign on scenarios, i.e. allowing a user to authenticate once but have simultaneous access to all the different application & tools developed across the federation.

The ESG-F Apache security filter inspects each incoming HTTP request for the presence of a security token, the absence of such a token or the presence of an expired token triggers an authentication challenge, i.e the request is rejected until the client successfully authenticates.

The ESG-F security infrastructure supports two modes of authentication: X509 Certificates for machine to machine communication scenarios; OpenID for human to machine communication scenarios. In order to support the former the ESG-F hosts a certificate authority server, in order to support the latter the ESG-F supports an array of OpenID providers.

The ESG-F security Apache filter is configured so as to protect the web application under its domain of responsibility, in this case the Metafor web services. Low level implementation details can be found by following the relevant links in the bibliography.

7. REST Controllers



Requests that pass the Apache filters are routed to REST controllers for further processing. REST controllers are implemented in the Python programming language and are hosted within the Metafor web service Pylons application. A REST controller manages access to a resource, i.e. an informational unit. REST controllers fulfil their responsibilities by orchestrating calls to the component layer below. Whilst the components perform low level tasks, it is the controller that decides which tasks are to be performed

Several types of resource are supported by the Metafor web services. Each type of resource supports a different use case, sometimes a resource may support multiple use cases. Some resources are read only, in this case the corresponding REST controller raises an error if HTTP operations other than GET are invoked.

Document

A document is a CIM compliant XML document/element.

Use Case

At various points within the climate modelling process an institute generates metadata, the document REST controller supports the persistence of this metadata. Institutes create instances of CIM compliant documents & send them to the service for persistence within a repository. The document can then subsequently be retrieved, deleted or updated. Updates cause the document version to be incremented.

Implementation

The document service is implemented as an AtomPub service. The service supports filtered feeds and the standard REST operations of create, retrieve, update & delete. The content of incoming create & update messages are passed to a validation component for validation, validation failure results in the termination of request processing. The service interacts with a data access component in order to communicate with a back-end database in which the documents are stored.

Technical Notes

1. The controller is implemented using a Python plugin called Amplee which permits a controller to act as an AtomPub endpoint.
2. All incoming AtomPub messages are validated by a validation component.
3. The back-end database system that stores all CIM documents is a specialized XML database called eXist. A data access component exposes functions for interacting with the eXist database. The document service uses this component to perform all database related work.
4. The Atom feeds list the CIM documents as URL's, the client navigates to the URL in order to download the actual CIM document.

REST Endpoints

URI	**/repository/document**
Resource	A newly created CIM document
HTTP Verbs	POST
Mime Types	XML

URI	**/repository/document/[document-uid]**
Resource	An individual CIM document identified by a uid (universally unique identifier).

HTTP Verbs	GET, POST, PUT, DELETE
Mime Types	XML

URI	**/repository/document/[document-uid]/[document-version]**
Resource	An individual CIM document identified by a uid and a version.
HTTP Verbs	GET, POST, PUT, DELETE
Mime Types	XML

Query

Queries are searches executed against the backend CIM database.

Use Case

A diverse community, ranging from institutes to students to researchers to private companies to policy makers, augment their decision making processes by querying the CIM repository. Whilst each user group has differing query requirements, a constrained set of queries supports most if not all requirements.

Query tools, developed either by Metafor or by 3rd parties, leverage the QS to execute queries:

- *Basic* – basic keyword query over all CIM documents.
- *Advanced Data* – advanced constrained query over CIM data documents.
- *Advanced Model* – advanced constrained query over CIM model documents.
- *Advanced Simulation* – advanced constrained query over CIM simulation documents.
- *Advanced Experiment* – advanced constrained query over CIM experiment documents.

Queries are driven by criteria, i.e. a set of fields that a user can populate. For each type of query a set of results are returned based upon the populated criteria.

Implementation

The query service supports two resources:

1. Criteria are XHTML forms that can be displayed to a user. Once the user has populated the criteria, the XHTML form is submitted in order to retrieve the query results..
2. Results are a set of records that match the submitted criteria. The service validates the incoming criteria using a validation component. Valid criteria are passed to a data access component that executes a query against a back-end database and returns an XML document summarising the query details and each CIM document that matched the query. These results are passed to a formatter and returned as an XHTML table. A balance needs to be struck between returning too little summary information which would make it difficult for users to interpret the results and returning too much summary information (even returning a complete CIM document) which would make the web service inefficient as well as making it unlikely that the formatter could render it usefully.

Technical Notes

1. The controllers are implemented as standard Pylon REST controllers.
2. All submitted XHTML criteria forms are validated by a validation component.
3. The back-end database system against which the queries are executed is is a specialized XML database called eXist. A data access component exposes functions for interacting with the eXist database. The query service uses this component to perform all database related work.
4. The query results are summaries of matching CIM documents constrained to a maximum of 1000 records so as to limit bandwidth usage. The results also include statistical information such as duration of query.
5. The set of XML "facets" that ought to be returned for each query type can be specified in an XML configuration file which forms part of the eXist back-end database. This configuration file can be easily edited by developers without having to redesign any of the XQuery code driving the actual query.

REST Endpoints

URI	**/query/criteria**

Description	Links to supported query criteria.
HTTP Operations	GET
Mime Types	XML

URI	**/query/basic/criteria**
Description	Search criteria used to drive the basic query.
HTTP Operations	GET
Mime Types	XHTML

URI	**/query/advanced/data/criteria**
Description	Search criteria used to drive the advanced data query.
HTTP Operations	GET
Mime Types	XHTML

URI	**/query/advanced/experiment/criteria**
Description	Search criteria used to drive the advanced experiment query.
HTTP Operations	GET
Mime Types	XHTML

URI	**/query/advanced/model/criteria**
Description	Search criteria used to drive the advanced model query.
HTTP Operations	GET
Mime Types	XHTML

URI	**/query/advanced/simulation/criteria**
Description	Search criteria used to drive the advanced simulation query.
HTTP Operations	GET
Mime Types	XHTML

URI	**/query/results**
Description	Links to supported query results with default query parameters.
HTTP Operations	GET
Mime Types	XML

URI	**/query/basic/results**
Description	Search results returned from basic query execution.
HTTP Operations	GET
Mime Types	XML, XHTML

URI	**/query/advanced/data/results**
Description	Search results returned from advanced data query execution.
HTTP Operations	GET

Mime Types	XML, XHTML

URI	**/query/advanced/experiment/results**
Description	Search results returned from advanced experiment query execution.
HTTP Operations	GET
Mime Types	XML, XHTML

URI	**/query/advanced/model/results**
Description	Search results returned from advanced model query execution.
HTTP Operations	GET
Mime Types	XML, XHTML

URI	**/query/advanced/simulation/results**
Description	Search results returned from advanced simulation query execution.
HTTP Operations	GET
Mime Types	XML, XHTML

Validation Report

Validation reports are generated against potential CIM documents.

Use Case

Whether it is an institute that has automatically created a CIM document as part of its climate modelling workflow, or whether it is an individual who has hand crafted a CIM document, both actors will wish at some point to validate the generated CIM document in order to guarantee conformance to the CIM ontological schema. The service clients require more than a simple valid / invalid response, in fact they require detailed validation reports so as to quickly diagnose issues.

The validation report service supports this requirement by returning detailed reports regarding the validation status of the newly created CIM document. Such reports permit developers to quickly diagnose faults in their CIM document creation processes. An invalid CIM document will not be allowed to be archived into the eXist repository mentioned above.

Implementation

A received CIM document is passed to a validator component for validation. The validator component executes a series of validators against the document and returns a validation report based upon the errors reported by the validators. The following validators are executed against a received CIM document:

1. Syntactic Validator - Verifies that the CIM document is well-formed XML & conforms to the CIM XML schemas.
2. Semantic Validator - Performs various types of validation such as ensuring that all of the XLink references embedded within the CIM document point to valid units of metadata. These sorts of checks are encoded using the Schematron language.
3. Controlled Vocabulary Validator – Verifies that the series of Controlled Vocabularies used by the CIM document makes use of a valid set of Controlled Vocabularies (CVs). A CV defines a set of terms that can be used in particular locations with a CIM document. It is usually governed separately from the CIM and evolves at its own rate. For example, the CMIP5 CV will consist of terms like "atmosphere", "ocean", "sea-ice", "biogeochemistry", etc. to be used as the content of the CIM XML element "modelType."

Technical Notes

1. The controller is implemented as a standard Pylon REST controller.

2. All XML validation operations are implemented using a Python XML plugin called lxml.

REST Endpoints

URI	**/validation/report**
Description	Validation report generated against newly created CIM document.
HTTP Operations	POST (message content must be a CIM XML document)
Mime Types	XML, JSON, XHTML

Controlled Vocabulary

A Controlled Vocabulary (CV) is a collection of vocabulary terms that can form the content of particular CIM XML elements or attributes and is governed separately from the CIM.

Use Case

CIM documents need to specify the CVs that it will be using. Different user communities may use different CVs. For example, an impacts community may have one set of terms they use when describing component models, while an ensembles community may use another set. Those terms should not be hard-coded into the CIM, rather the impacts and ensembles communities each govern their own terms. Then, during validation, those terms need to be automatically retrieved from the CVS and checked against the content of the CIM document being validated.

Implementation

The format of a CV will be XML that adheres to the SKOS (Simple Knowledge Organization System) format. The Metafor CV will be implemented in collaboration with the BODC (British Oceanic Data Centre) who already host a Controlled Vocabulary service. This service allows for the updating of the CV as per the governance rules. Such details are beyond the scope of Metafor.

However, in order to be used alongside the CIM XML Schemas, a CV XML file will need to be translated to Schematron. A CIM document element that is bound to a particular CV will need to specify the CV details. Using those details, the CV can be retrieved from the BODC service, translated to Schematron, and the CIM document can be validated against the Schematron rules

Tracking

Permits cross referencing of CIM documents with meta-data held in 3rd party repositories.

Use Case

The information described by and referenced by CIM documents may be archived in other (ie: non-Metafor) repositories. In their native format they will obviously use native identifiers. The tracking service provides a mapping from native identifiers to CIM identifiers.

These native identifiers can also be used when a group of self-referential CIM documents are being constructed. For example, a set of coupled model components can be constructed together before any of them have been archived into the CIM repository. Without being archived, they will not have a CIM identifier. However, they will naturally have a native identifier (the id or name used by the institute developing and/or documenting the components).

Implementation

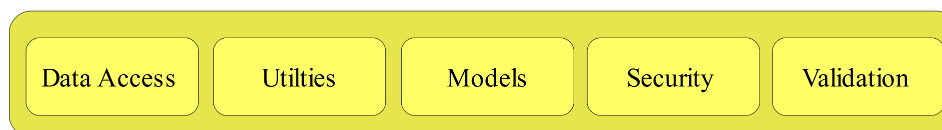
The tracking service will be implemented as a simple lookup table stored in a database.

REST Endpoints

URI	/tracking/[native-id]
Description	returns the CIM identifier that corresponds to the native id provided.
HTTP Operations	POST

Mime Types	XML

8. Components



Components perform low level tasks such as communicating with databases, opening files, representing data structures...etc. Components are developed using the Python programming language and are unit testable in isolation from the controllers. For each component a unit test module is also written so as to automatically test component functionality, this technique guarantees robustness.

Security

The security component encapsulates the validation of all incoming HTTP requests . HTTP requests are validated in order to guard against attacks such as cross-site scripting (XSS) & SQL injection. Such validation involves applying rulesets against each request, a ruleset is a collection of rules chained together.

Data Access

The data access component encapsulates communication with backend database servers. Two types of database are used by the Metafor web services: eXist and PostGres. All communication with either database is via a native query language, for eXist see XQuery, for PostGres see T-SQL.

The component exposes a function for each supported database command. The function implementation details differ according to database type, however at a high level they are identical:

1. Open a connection;
2. Prepare the command to be executed;
3. Execute the command;
4. Close the connection;.

For further details of how to execute database commands see *Layer 4 - Resource Managers*.

Validation

The validation component encapsulates the validation of validation of CIM documents. CIM documents must be validated in order to guarantee the quality of meta-data held within the underlying repository. CIM document validation is detailed in the Validation Service technical notes above.

Models

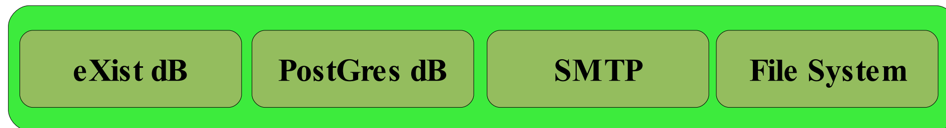
The models component simply contains models (i.e. data structures) that can be shared amongst components and/or controllers. Models are categorised according to function, for example there is a collection of models related to metrics gathering.

If a model has a matching representation in a relational database (e.g. PostGres) then the model is implemented using a 3rd party Python plugin called Elixir, otherwise it is implemented as a simple Python class. In either case it is possible to serialise/de-serialise a model instance to/from data formats such as XML & JSON.

Utilities

The utilities component is a non-specialized component that simply contains a set of utility functions used in miscellaneous scenarios such as sending an email, opening a file ...etc. It is likely to be invoked from many different controllers/components.

9. Resource Managers



eXist dB

CIM documents are stored in the open-source native XML database eXist. Because CIM documents are themselves XML documents, using an XML database is a natural fit; no ORM is required. eXist can be manipulated using the XQuery language. It comes with a built-in REST HTTP interface. Each command maps to a unique local URL. XQuery parameters are passed as name/value pairs to the HTTP request. XQuery results are XML documents. The eXist resource manager provides XQuery commands to retrieve, store, update, and query CIM instances from the repository.

PostGres dB

Execution of T-SQL commands against the PostGres database involves using a Python based Object Relational Mapping (ORM) library called SQLAlchemy. SQLAlchemy is simple to use and supports the mapping of entities, i.e. Python data structures, to relational database tables, i.e. PostGres tables. SQLAlchemy automatically generates the required T-SQL.

SMTP

Certain use cases require notification to be sent by the system in the form of emails. An SMTP server is necessary for supporting this, such a server is hosted within by the BADC. All SMTP operations are implemented using the standard smtplib Python library.

File System

The system requires files to be stored locally, such files include code files, configuration files, XML Schema files ...etc. The local file system provide such storage functionality. The in-built io Python library is used for all file I/O operations.

10. Bibliography (links valid as of 01st August 2010)

Metafor	Overview	http://metaforclimate.eu
	Deliverables	http://metaforclimate.eu/trac/wiki/deliverables
	CIM	http://metaforclimate.eu/trac/browser/CIM/trunk
SOA	Overview	http://en.wikipedia.org/wiki/Service-oriented_architecture
	HTTP 1.1	http://www.w3.org/Protocols/rfc2616/rfc2616.html
	Web Services	http://en.wikipedia.org/wiki/Web_service
REST	Overview	http://en.wikipedia.org/wiki/Representational_State_Transfer
	Atom Publishing	http://bitworking.org/projects/atom/rfc5023.html
	XML encoding	http://en.wikipedia.org/wiki/XML
	JSON encoding	http://www.json.org/
	Base64 encoding	http://en.wikipedia.org/wiki/Base64
Security	PKI	http://en.wikipedia.org/wiki/Public_key_infrastructure
	X509 Certificates	http://en.wikipedia.org/wiki/X.509
	OpenID	http://openid.net/
Hosting	Linux	http://www.linux.org/
	Apache	http://www.apache.org/
Development	Pylons	http://pylonshq.com/
	Python	http://www.python.org/
	Schematron	http://www.schematron.com
	SKOS	http://www.w3.org/2004/02/skos/
Databases	eXist	http://exist.sourceforge.net/
	XQuery	http://www.w3.org/TR/xquery/
	PostGres	http://www.postgresql.org/
	T-SQL	http://www.tsql.de/transact-sql