Live Model Pointers
A requirement for future model repositories

Keith Duddy
QUT/Smart Services CRC

8 April 2009

1 Introduction

Model interoperability is a topic that assumes that models are created and transported between several tools in a tool chain used to create some end product. Most of the time that product is a software artifact such as an application or service. This paper asserts that model import/export between tools is insufficient to avoid the data decay problems that can arise when the source of a model element, and its ultimate usage are disconnected.

The case used to illustrate the idea of *Live Model Pointers* (LMPs) is a Service Broker which stores descriptions of services (both software and human based) according to a number perspectives which provide the full business context of the deployment the services. An overview of the Broker is provided, followed by an approach to synchronisation of information in the Broker (model elements) with definitive sources of that data.

A sketch is then provided of the concept of LMPs, and their implementation. A case is made for the concept of LMPs to be incorporated into future modelling frameworks and repositories.

2 Service Broker as a Model Repository Case Study

The Smart Services CRC's Service Delivery Framework project is currently engaged in designing and building a Service Broker. The broker's role is to provide a repository of services offered in a particular company or marketplace for discovery by service clients. The services can be described for the purposes of discovering:

- Service design information
- Service pricing and legal information
- Service deployment context
- Service provider and consumer profiles

- Service technical interface description
- Service access control, metering and billing, and other kinds of mediation

There are three core perspectives for metadata about services that are kept in the broker: Organisational, Technical, and Operational. These aspects are modelled using the eMOF meta-metamodel in EMF, and additional representations of the model are derived for various purposes – UML (visual), XML Schema (model transfer) and Hibernate (database creation). The metamodel of services is known as the Universal Service Description Language, USDL, for historical reasons. Additional perspectives of services may also be mixed in using a model perspective navigation framework. For example, in a banking scenario the three core perspectives may be augmented by a Financial perspective, and in a government lands management scenario there may be an additional Geospatial perspective.

2.1 Source of Truth

The information the broker seeks to expose to service clients is largely a duplication of information that is stored in various other places within the company or marketplace in which it is deployed. Some notable cases of this are:

Organisational Information. The broker intends to provide the prospective service client with the usual kinds of name, address, web page, contact person and email address information that one would expect to find in an old-fashioned yellow pages (the original service broker). However, the place in which an organisation usually manages and keeps this kind of information up to date is in a CRM system.

Technical Interface Information. The broker is agnostic about what kinds of service implementation technologies are used (and also represents services which are only provided by human beings). Therefore it represents interfaces as summaries of the major operations that are available through a service, and their inputs and outputs. However, this information should rightly be derived from and kept consistent with other sources, such as UML, WSDL, Java Classes, IDL etc.

Legal Information The broker attempts to give the whole context of service usage, including the legal framework for it. Therefore contracts would ideally be used as the source of information for the legal clauses exposed in the broker.

Service Level Part of the legal framework for the provision of services in many cases is a Service Level Agreement, which specifies when the service is available, and with what reliability, time response, and other non-functional characteristics. The information in the broker should ideally be automatically derived from an SLA (or the SLA and the broker information derived from a model of service levels).

2.2 Repository Data Decay

The explanation given to this researcher about why previous service repository efforts have failed in corporate settings is that entries in such repositories were created only at the beginning of the service life cycle, and never updated. In addition, maintaining such a repository was left to the force of will of a particular employee, who was given this task on top of their other duties.

The result was that the service descriptions in the repository became out of synch with the actual deployed services both because the service owner had not tasked anyone with keeping the repository up to date, and it was too difficult or time consuming for the repository maintainer to poll all the services to check for updates. The clients of service repositories then realise over time that the entries in the repository contain stale data, and stop using it as a definitive source of information.

3 Models by Reference

In a utopian world, a single metamodelling language is used inside an enterprise to allow for tool chains and model repositories to seamlessly create, maintain, and transform models across domains and layers of abstraction. In this utopia keeping references between models would be as easy as it currently is to keep foreign keys to database tables in the same RDBMS.

However, in the real world, "models" are maintained in many databases, directories, files and repositories. And this means that references between models are kept in many ad-hoc ways, usually by capturing a primary identifier of a concept in a different modelling framework, and replying on users to cut and paste IDs between UIs, or create applications that effectively do the same. Or in many cases, by copying the data from one format into another, with no ability to keep consistency between them.

3.1 The Next Best Thing

In the absence of a unifying modelling framework with the ability to keep pointers between all relevant models, the next generation of model repository will need to be able to attach small model query components, which we call *Live Model Pointers*, to certain classes in models which allow them to go outside the repository, and draw in the information that is kept in a definitive form elsewhere.

To return to our motivating Service Broker case study, the first case given above is of an Organisation class which represents a service consumer. The definitive information about Organisations is kept in a Customer Relationship Management (CRM) system. Let's use the example of the popular Software as a Service CRM solution, salesforce.com, which provides customer data both via a customer web UI, and Web Services access via WSDL. So rather than copy the data from the CRM at the time the organisation first becomes a client of a service, we would attach an LMP to the Organisation class in the next-gen modelling repository which is Web Services enabled, and the Live Pointer would contain the following elements:

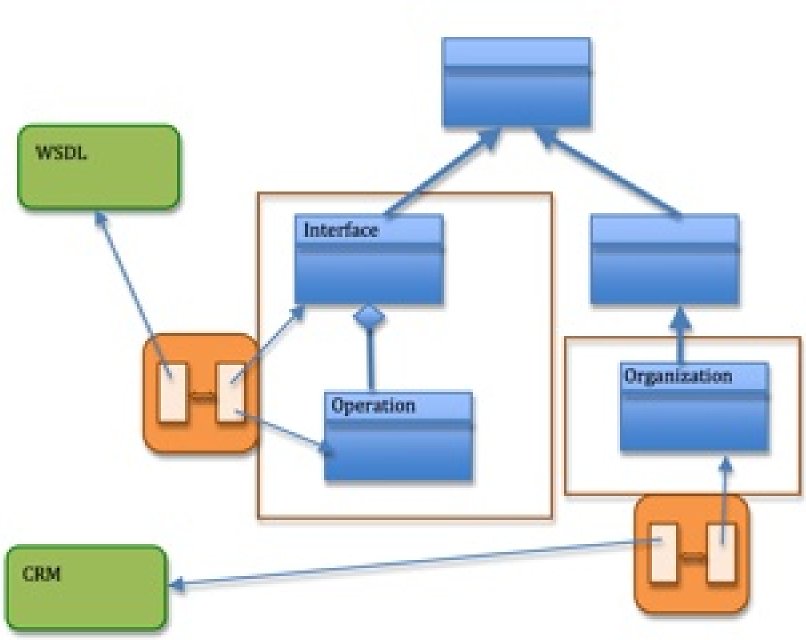

Figure 1: Model with Live Pointers

- A nominated set of attributes in the model that form the "foreign key" of the organisation in the salesforce.com
- A URI on which to connect to salesforce.com via Web Services.
- A Web Services client that is capable of extracting the key from the model, and doing a lookup on the organisation's details at salesforce.com.
- A Mapping from elements in the returned XML result to the remaining non-key organisation attributes in the Broker.
- A number of policy settings: caching, timeouts, etc

Another example is the Interfaces and Operations classes of the USDL model. These represent only the names of the major access mechanisms for the service, in terms of a group of named operations in an interface. In the case where an interface is a WSDL specification, the corresponding information is a set of PortTypes and their contained Operations. The broker also contains the URI of the WSDL specification in case the Broker's client wishes to look up the details. If we wrap the Interface and Operation classes in our next-gen repository by an LMP, it would contain the following elements:

- An attribute in the model that contains the URI for the WSDL as deployed on the Web.
- An HTTP client that is capable of retrieving the WSDL file from its Web server.

- A Mapping from elements in the returned WSDL (XML) to extract the Interface/Operation groupings into the relevant model elements and their attributes the Broker.
- A number of policy settings: caching, timeouts, etc

3.2 LMPs in the Abstract

So let's generalise. LMPs need:

- A specification of the source of the data, in terms of a connection point or other resource locator
- Key attribute(s) to use in a query at that connection point (or on a query to some artifact retrieved from that connection point).
- A remote (or local) access mechanism to connect to the connection point using the correct protocol.
- A transformation from data returned from the query or lookup to the connection point into the model elements that the LPM wraps.
- A number of policy settings: caching, timeouts, etc

An optional extra is an event listener that receives notifications that a remote data source has been updated, allowing the next-gen repository to update its cache.

3.3 Sources of LMP data

In the general case, the Mapping part of a LMP would allow for rich model-to-model transformations when the pointer is to models in the same repository framework. But many kinds of mapping tools would be necessary, depending on the kind of data source. The next-gen model repository would come with LMP adapters that allow model elements to be populated from the following kinds of sources:

- XML files and databases
- MOF repositories model files
- UML repositories and model files
- LDAP and X.500 Directories
- Customer Relationship Management (CRM) systems
- IDEs
- Relational Databases (RDBMS)
- RDBMS Abstractions (e.g. Hibernate)

- Structured Text Files (e.g. Word 2007 XML format)
- Unstructured Text Files (e.g. PDF)
- ...

4 Conclusion

In our case study of a Service Broker repository, the models representing the majority of elements of services are replications of data from other, more definitive, sources. A major cause of failure of service repositories in organisations is the slow decay of the replicated parts of the service metadata with respect to their definitive sources, until a tipping point is reached, and the clients of the repository no longer trust it to contain up to date information. The solution to the problem is a repository enabled with Live Model Pointers that contain: a locator for the external information source; a nominated key to the external data; a remote query mechanism to retrieve the remote data from its source; and a parser/transformer that can extract the relevant information into the attributes of the model elements that the LMP wraps.