## AccelRT Service Invocation Design

### Introduction

This document describes the design of a general service invocation architecture for AccelRT.

The requirement is to be able to register an arbitrarily long list of services that are available either as functions invoked from the UI or as jobs triggered from a time based process.

The registration data will enable the calculation of a URI that invokes a REST based web service in an asynchronous fashion.

Many web services will return data to a particular AccelRT endpoint to have the data properly placed in the AccelRT data model. However, simple web services may optionally return data to an identity-attribute-value (IAV) or *triple* table to store data not in the current data model. There will be a REST service endpoint to receive triples together with associated metadata that allows the returned value to be displayed at the correct point.

Compliant web services will return progress messages through a standard interface so that a user will be able to see the progress of a call. These will be triples of the type ‘message’. Services will identify a particular invocation with a persistent identifier so that all messages for a particular job can be grouped together, and deleted once completed or cancelled.

### Design overview

#### Service invocation

Services will be registered with the AccelRT database. Each service will have a base URL, a human readable name and a description. A service can be associated with zero-to-many web pages or views: for each web page that may call the function the database will maintain the text of the button used to invoke it, a list of parameters required by it and the associated local variable names that allow a standard Python routine to fill parameters with appropriate values. The service call will thus be calculated as the combination of the base URL, a ‘?’ and then *param(1)*=*value(1)*& *… param(n)*=*value(n)*

This metadata will allow a webpage to work out which web service functions to display, how to display them and how to invoke them.

For example, on the ‘treatment\_plan’ webpage we would like a button with the text ‘total dose’ that submits the current treatment plan to a web service *http://accelTR.org.uk/total\_dose\_calculation* so that the total dose can be calculated. This would require the following rows in tables in the database.

When a service is invoked, it must maintain a unique identifier for the invocation so that returned values and messages can be properly grouped when returned to the database.

|  |  |  |
| --- | --- | --- |
| tblService | | |
| url | name | description |
| http://accelRT.org/total\_dose\_calculation | Total Dose | A web service that calculates the total planned dose for a radiotherapy treatment plan |

|  |  |
| --- | --- |
| tblServicePageOrView | |
| Service | PageOrView |
| http://accelRT.org/total\_dose\_calculation | treatment\_plan |

|  |  |  |
| --- | --- | --- |
| tblPageOrView | | |
| name | button name | type |
| treatment\_plan | calculate total dose | page |

|  |  |  |
| --- | --- | --- |
| tblParameter | | |
| PageOrView | parameter | value |
| treatment\_plan | plan | pyPlan.plan |

Where the named item is a view, the parameters will be matched with the name of the column in the view that supplies the value. Thus it will be possible for a chron job to open that view into a cursor and invoke the service for each row returned by the view.

#### IAV Return

Where there is no specific place in the current data model for returned data, it may be added to a triple table that identifies a particular entity in the database with which an attribute and a value. In the above example, we would expect the service to return a *total dose* in Gray for a particular treatment plan. Thus the IAV service must accept a set of parameters:

* Service\_URL: the service called to return the attribute value pair
* Invocation\_ID: a unique identifier for the particular invocation of the service – here *http://accelRT.org/total\_dose\_calculation*
* Entity\_type: the table name that stores rows describing the entity for which the attribute applies – here *tblTreatmentPlan*
* Entity\_identifier: the identifier of the particular entity in that table for which the attribute-value pair describe
* Attribute: the name of the attribute returned – *Total Dose*
* Value: the value of the attribute – for example *50 Gy*