5-1-2010

Designing An Ajax-Based Web Application Restfully

Benjamin Daggolu

DESIGNING AN AJAX-BASED WEB APPLICATION RESTFULLY

By

Benjamin Daggolu

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

May 2010

DESIGNING AN AJAX-BASED WEB APPLICATION RESTFULLY

By

Benjamin Daggolu

Approved:

Tomasz Haupt Associate Research Professor of Computer Science and Engineering (Director of Thesis)	Edward B. Allen Associate Professor of Computer Science and Engineering and Graduate Coordinator (Major Professor)
Thomas Phillip Professor of Computer Science and Engineering (Committee Member)	Sarah Rajala The Dean of Bagley College of Engineering

Name: Benjamin Daggolu

Date of Degree: May 1, 2010

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Edward B. Allen

Title of Study: DESIGNING AN AJAX-BASED APPLICATION RESTFULLY

Pages in Study: 94

Candidate for Degree of Master of Science

The development of an AJAX-based web application involves several challenges as the webpage is updated by using the AJAX calls without reloading the entire page as in any traditional webpage. This prevents one from going back to the previous view of the page as the browser does not reload the entire page; instead it only updates the page. My hypothesis is that if an AJAX-based application is designed by using the software architecture style called the Representational State Transfer (REST), then it is possible to overcome these challenges, which cannot be handled by using web-services. In order to investigate this, the Material Properties Repository, an AJAX-based application was redesigned by using REST. The results support my initial hypothesis. In this process of designing MPR using REST, a generalized software engineering process was created for designing an AJAX-based application RESTfully.

ACKNOWLEDGMENTS

I would like to thank Dr. Tomasz Haupt for directing my thesis and teaching me several software engineering concepts. I would like to thank Dr. Edward B. Allen for being my major advisor and for his suggestions throughout my graduate program. I would like to thank Dr. Thomas Phillip for being on my graduate program committee and evaluating my thesis work. I would to thank the Department of Energy (Grant ID: DOE 06100904) for funding me to work on the Material Properties Repository. I would like to specially thank Dr. Mark F. Horstemeyer for giving me this opportunity to work at the Center for Advanced Vehicular Systems (CAVS) and for his encouragement and support. I would also like to thank the Empirical Software Engineering group for their valuable feedback on my research. I would like to thank God for everything and my family and friends for their love and support.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

This thesis presents the software engineering process for designing an AJAX (Asynchronous JavaScript and XML) based web application using the software architecture style, Representational State Transfer (REST). The motivation for this research is the challenges faced while developing the AJAX-based application called the Material Properties Repository (MPR) at the Center for Advanced Vehicular Systems (CAVS), MSU [1].

1.1 What is the Material Properties Repository (MPR)?

1.1.1 Functionality

The Material Properties Repository integrates three different web applications. They are [1]:

A. Repository of experimental data

B. Repository of material constants

C. Online model calibration tools

Repository of experiment data: The MPR allows users to upload Experimental Data to the repository through the upload tab. The experimental data come from physical measurements of materials' properties. Currently, the repository supports stress-strain data, images of the microstructure, and strain-life (fatigue) data. The repository of experimental data is the database of the results of measurements (often requiring transformation of a raw data, such as deriving true stress-strain from force-displacement), or data taken from the literature. The user can upload the data to the repository, search for a particular data set, and retrieve the data for further analysis – typically to derive material constants. The Experimental Data can also be uploaded from the tools directly. The data sets from the repository can be viewed from either the 'Experimental Data' tab or the 'Materials' tab of the MPR webpage. The Experimental Data can be viewed as metadata, a plot, or it can be opened in a tool.

Repository of material constants: The MPR provides a repository to store the material constants calibrated from the experimental data. For the purpose of numerical simulations, a material is represented by mathematical models, such as the Damage Model, that provide a prediction of the material behavior subjected to certain conditions. The models are parameterized by a model-specific number of constants, referred to as material constants. The repository of material constants is a database of these constants. The constants are derived from the experimental data. The user can upload the constants to the repository, search for the constants of a particular model of a particular material, and retrieve the constants for further analysis – typically to use them in numerical simulations, such as finite element analysis using Abacus, LS-Dyna, or other software. These constants can be viewed from either the 'Material Models' tab or the 'Materials' tab of the MPR webpage. The material constants can be viewed as either metadata or in a tool. The ‘Materials’ tab of the MPR page displays both the material constants and the experimental data simultaneously.

Online Model calibration tools: The model calibration is the process of deriving the material constants from the experimental data, usually by performing a fit of a model-specific function(s) to the experimental data. MPR provides online tools to perform the models’ calibrations. Currently there are three tools available in MPR; they are Damage Fit, Image Analyzer, and Multistage Fatigue Fit. They can be accessed from the ‘Tools’ tab of the MPR page. These tools allow the user to extract model-specific material constants from the experimental data. These tools also allow the upload and download of experimental data and saving of the calibrated material constants.

1.1.2 Integration

Although each of the aforementioned web applications can be used independently, the advantage of MPR is that MPR integrates all three applications into one, thus allowing the complete cycle of analysis: uploading experimental data, applying the calibration tools to extract the material constants, saving the constant to the database, and retrieving them in a form suitable to perform numerical simulations.

1.2 Hypothesis

The hypothesis of our research is as follows:

Incorporating REST methodology into the design of an AJAX-based web application that aggregates (mashes up) disparate services will lead to an efficient, easy to maintain, and extensible implementation and efficient GUI design.

The Material Properties Repository is a web application, which provides access to a number of remote services. However, by applying AJAX technology that enables Rich User Interface and services mash-ups, this distributed system has the look and feel of a stateful desktop application.

Developing such a web application using web services with SOAP has become very complicated and lead to a proliferation of objects implementing the rich functionality of the application and therefore it unavoidably results in an unmanageably complex interface (in terms of number of methods) and a confusing GUI. As a consequence, the number of lines of code unnecessarily increases, and the risk of code duplication increases. The code maintainability becomes poor, and it is difficult to track the state of the application and perform corrective actions, if an exception should occur.
The goal of our research is to prove that building a web application RESTfully, that is, with design focus on resources and transitions between the representations of these resources as opposed to focus on methods operating on the remote objects, will make the system efficient, easy to maintain, and extendable. The uncertainty of our approach is that this approach has been never tried on AJAX-based applications.

1.3 Research plan

1.3.1 The Problem

The front-end of the repository of experimental data and the repository of material constants is integrated into one application as MPR, and each of the tools are implemented separately which can be navigated through the MPR. Each of those pages serves as a Graphical User Interface (GUI). The GUIs allows the user to interactively perform all actions supported by MPR by utilizing web widgets on the pages. GUIs are implemented using Asynchronous JavaScript and XML (AJAX). The processing of the actions associated with GUI widgets is processed either locally using JavaScript, or it delegated to the server side through AJAX calls. Regardless of whether the processing is local or remote, only a relevant fragment of the web page hosting GUI is updated; indeed, there is never a need to reload the whole page, unless a model calibration tool is invoked. By employing AJAX, the GUI has the look and feel of a stand-alone application with a very rich interface.

There are, however, three major disadvantages of using AJAX. AJAX invites an RMI-style implementation: each AJAX call invokes a particular method (“action”) to be performed. This “action” is typically implemented as a dedicated Java Server Page (JSP) servicing the call which returns a static or dynamically generated html fragment to be “pasted” to the GUI page at predefined location. The dynamical generation of the JSP response is performed either by the JSP page itself (or JavaBean), or by invoking an operation of the Web Services (Data Service or Compute Service). The rich functionality of the GUI typically results in a proliferation of the number of distinct methods (actions and/or JSP pages) to be implemented that complicates the service interfaces, making them difficult to maintain and extend.

The GUI behaves like a stand-alone application, contrary to the user experience with web-based applications that take the advantage of the web browser functionality, such as back-button, or creating a bookmark, so the user can return to a previous “state” of the browser. The problem is that because the GUI is implemented as a single page, from the browser point of view the "state" of the application does not change (it is still the same page defined by the GUI URL).

Finally, the GUI is stateless, it is complex, and it requires applying non-web GUI design. This is counterintuitive for both the developers and the users. Indeed, the first implementation of the Remote Method Invocation (RMI) GUI proved to be confusing for the users.

The current implementation of the Material Properties Repository does not follow the REST style; hence it has the above cited problem. Our assumption is that we can overcome the above problems by redesigning the MPR using REST principles.

The typical software engineering process for the developing a RESTful application is given in section-2.6.4. But this process is used only to develop a RESTful application that does not involve AJAX and mashup. So our research plan is to propose a Software engineering process for developing a RESTful application that involves AJAX along with mashing-up of web-services.

Goal-1: Software engineering process to design a RESTful application that involves AJAX and mashup.

AJAX challenges:

- The resource URI is not the same as the page URL. We have a single page URL that "aggregates" multiple resources. The state of the client application (Web Browsers) must be captured differently, so that each state can be reproduced at will.
- The richness of an AJAX GUI "supersedes" the hyperlinks embedded in the resource representation. The state transition is thus not achieved by requesting

a new resource associated with the URI of the hyperlink; instead the state transition is triggered by a JavaScript action associated with a widget.

Goal-2: It should be easy to add new resources (e.g., new data types) while performing "standard" operations such as show metadata, show plot, download, upload, etc.

In the current design, in order to add a new data type the code needs to be changed in several location, our goal is to make it centralized so that new data types should be added at once location and it should be populated to all the locations where it is used and the new design should perform all the necessary featured of the current design.

Goal-3: It should be possible to reproduce any state of the application at will.

In the current implementation it is not possible to go back to the previous view of the webpage i.e. bookmarking and back button features are not available. Our goal is to implement them in the new design.

1.3.2 Identify the metrics of success

Finite state machine (FSM): FSM can be used to verify the completeness and efficiency of the Material Properties Repository. The FSM can be considered a graph that consists of nodes (states) and the transition arrows which represent the change of the states. Each module of the MPR can be considered as a state of the FSM and the transitions are the events of the user or the internal events. The completeness and the efficiency of the system can be determined by establishing a finite state machine that can be verified with the requirements of the system. The ease of adding new features to the system can be determined by the amount of changes that have to be made to the FSM in order to accommodate the new feature. We can say that the system is built successfully if the FSM has all the features of the system which are specified in the requirements and the correct transitions among them.

CHAPTER II

RELATED WORK

2.1 Web 2.0

The introduction of Web 2.0 has promoted the web-services to the next level, by providing an interactive and collective collaboration that provided new features for both the developers and the users. The introduction of the Web 2.0 gave rise to some of the highly successful social applications like YouTube, Facebook, Orkut, Flickr etc. Web 2.0 is a collection of technologies along with business strategies and social trends. Unlike Web 1.0, web 2.0 not only allows accessing and contributing to the contents of the web, but also allows the user to keep track of the latest contents of the webpage without even accessing the page with the help of Really Simple Syndication (RSS) and ATOM. It also provides a easy way of developing web applications that aggregate, different information or services from the internet [2].

Web 2.0 provides several key technologies like Blogs, Really Simple Syndication, Wikis, and Mashups [2].

Blogs: A blog is a two way communication tool where users can post their views and comments, where these entries are displayed in reverse chronologically. Blog stands for Web log [2].

Really Simple Syndication (RSS): RSS is a web format used for the syndication of a webpage or a blog. It uses an XML file that links the information source and the information item in order to inform the subscriber about the updates [2].

Wikis: Wiki is a web-based content managed system that is maintained by collaborative authors. Any user can add, revise or delete the contents of a wiki page [2].

2.2 Mashups

Mashups are the important feature of the Web 2.0. A mashup combines data, information or services from several web services on the internet to form a single service. It is very easy to create a web mashup rather than to code the application from scratch. Example: We can create a mashup web application for searching for houses in a locality by combining the services of a classifieds web-service and Google maps. The resultant mashup application will give us a map along with the houses for sale in a locality. Mashups are created by using the Application Programming Interfaces (API) which provides users a way of interacting with other services or webpages. Example: The Google maps API lets a user mashup its services with any other application [2].

There are three main advantages of using mashups, they are:

1. Improvised user interface.
2. Value added information as a result of aggregation.
3. Value added information augmented with an enhanced user interface [2].

In spite of the several advantages of mashup, it has a few limitations like reliability, continued support of the individual services, scalability and security [2].

There are three principle approaches to develop a mashup. They are, Google Web Toolkit, Adobe Flex and AJAX (Asynchronous JavaScript and XML) [2].

Limitations of Service mashups:

- The individual services should always be available for the mashup to work correctly. The functionality and the data structure of the source cannot change; if it does then the mashup has to be modified [3].
- The output and the input formats of each of the service should be negotiable and consistent [2].

2.3 AJAX

Asynchronous JavaScript and XML (AJAX) provide a highly interactive and very responsive web application. It aggregates several technologies like JavaScript, XML (Extensible Markup Language), XHTML or HTML and cascading stylesheets (CSS) [2].

In general a web application exhibits limited interactivity and slower performance than that of a desktop application. But the main advantage of a web application developed using AJAX is that it works and acts like a desktop application. An application built on AJAX can modify or add new information to a page without reloading the entire page, whereas with the traditional webpages the user should wait for the entire page to be reloaded even for a small change on the page [4].

AJAX uses the Extensible Markup Language (XML) to communicate data between the server and the client (browser). XML defines a set of languages with structured data in online documentation which can be developed by anyone with a set of markup tags [4].

AJAX uses Asynchronous JavaScript which is responsible for making asynchronous calls to the server and fetches XML documents, which lets the application retrieve, and simultaneously update, the webpage without reloading the entire page. AJAX uses the XMLHttpRequest object to make HTTP requests and retrieve data quickly in the background without any visible interruptions to the user [4].

Some of the advantages of using AJAX are that it makes the application very responsive. It makes the application work fast as it minimizes the traffic by requesting only the needed information from the server. AJAX is platform independent and its component technologies are familiar to the developers. By using AJAX we can create a web-based version of any desktop application, which can be accessed over the web without any installation which saves time and resources [4].

The limitations of AJAX are as follows:

- The individual technologies of AJAX (JavaScript and XML) have been in use from a long time, but now they are used in new ways under AJAX. This could give rise to security Vulnerabilities [4].
- There is a difference in the implementation of JavaScript in different versions of browsers which should be addressed [4].
- AJAX applications are based on client/server technology. The clients of the distributed system need a centralized server to communicate, as the clients cannot respond to the HTTP requests themselves [5].
- Unlike in traditional web applications, the users are unsure about which type of event would cause the user to wait for the response [5]. On a webpage it is

difficult for a user to know, which event would make an AJAX call and how long he should wait for the response.

- AJAX payloads can only be small. The server cannot send a large XML response to the browser as it will cause a delay [5].

2.4 Web-services

A Web services are the software systems designed for machine-to-machine communication over a network. The interface of these communications is described by a machine-processable format such as Web Services Definition Language (WSDL). Any system can interact with the Web service in a manner as specified by its associated description using the Simple Object Access Protocol (SOAP) messages, which are typically conveyed by using the Hyper Text Transfer Protocol (HTTP) with an Extensible Markup Language (XML) serialization [6].

2.5 What is Representational State Transfer (REST)?

Representational State Transfer (REST) is a style of the software architecture for distributed hypermedia systems such as the World Wide Web. The REST principles were introduces by Roy Fielding, who is one of the principle authors of Hypertext Transfer Protocol (HTTP). REST is often used to describe any simple interface which transmits domain-specific data over HTTP without any additional messaging layer such as SOAP or Session tracking via HTTP cookies [7].

An important concept in REST is that everything is defined in terms of resources, each of which is referenced with a global identifier (e.g.: - Uniform Resource Identifier). In order to access these resources, components of the application is communicate by exchanging the representations of these resources. Any application can interact with a resource by knowing two things: the identifier of the resource and the action to be performed (like GET, POST). The application should also know the format of the information that is returned, which is typically a HTML, XML or JSON (Java Script Object Notation) document [7].

The key aspect of REST is the state and the representation of each architecture data element. Different types of data elements of REST are listed in Table 2.1. In REST, components transfer only the representation of a resource but not the actual resource itself. The representation can be any standard data type which is decided based on the recipient's specification and also the nature of the resource [8].

Table 2.1 Types of Data Elements

Data element	**Example**
Resource	Target of a hypertext reference
Resource identifier	URIs
Representation	HTML documents

2.5.1 REST vs. Web Services

REST is a relatively new technology when compared to SOAP based web services. Web Services give new specifications for developing internet applications, whereas REST uses the already existing protocols of HTTP to design the services. Anyone who understands HTTP and XML can create a RESTful application without the use of any toolkit. REST provides interface flexibility when compared to Web Services as the resources are accessed using URIs which are familiar. A client who wants to access the resource can use the HTTP GET method along with the URI to use the resource. By changing the URIs, a client can access different services. In SOAP based Web Services, the service requests are passed using SOAP, which the developers should be familiar with, in order to form the request and parse the results. The developers need a toolkit to write and parse SOAP messages. Therefore REST has an edge over SOAP-Web Services regarding the interface flexibility [9].

REST supports light bandwidth when compared to SOAP-Web Services; SOAP messages are relatively long when compared to REST requests, as SOAP requires an additional XML wrapper along with each request and result [9].

SOAP-Web Services have more security concerns when compared to REST. SOAP based Web Services use the HTTP POST method for all the requests whether it is to modify a service or to only access them. Whereas, REST uses GET to access the resources and PUT and DELETE to modify them. Therefore the administrator or firewall can differentiate which request will modify the resource and which request only accesses them, based on the HTTP method used. In SOAP based Web Services the administrator should go through the SOAP envelope to discern whether the request queries for data or modifies it, which is not built into most of the firewalls. The administrators should take special measures to tackle this problem to ensure network security which is a resource and time consuming job [9].

The REST architecture style is gaining a lot of popularity in recent years. Most of the Yahoo web services are RESTful while eBay and Amazon have web services in both REST and SOAP [10]. Simple REST interfaces along with a text response are sufficient to develop web services, which save a lot of effort and time. On the contrary, SOAP based Web Services would be helpful when a client needs to send a large amount of data along with a request as it can be included in the SOAP envelope, which is not favorable in REST as it is not a good practice to include a large amount of data within a URI. Both REST and web-services are stateless [9].

2.6 Major concepts of REST

2.6.1 Resource

The abstraction of any information which can be named is considered as a resource. Some of the examples of a resource are as follows [8]:

- A document
- Image
- Temporal services like time or weather that change frequently
- Collection of resources
- Non-virtual object like system admin.

The Web is comprised of resources. A resource is any item of interest; for example, a metadata record describing an experimental data set is a resource. The resource is identified by its Uniform Resource Identifier (URI). The user can retrieve a representation of the resource by connecting to the URI of the resource. The representation places the client application (e.g., Web Browser) in a state. A new representation (possibly of another resource) places the client application into a new state. The client application changes (transfers) state with each resource representation, hence the name Representational State Transfer. REST is not a standard but it uses standards like, HTTP, URI, XML/HTML/GIF/JPEG/etc (Resource Representations) and Text/xml, text/html, image/gif, image/jpeg, etc (MIME Types) [11].

Each resource is mapped to a set of entities, whose values may be the resource representations and resource identifiers. A resource can be either static or dynamic. We say that a resource is static if the values of the set of entities never change after its creation. Whereas, for a dynamic resource, the value set keeps changing with time. The distinction, whether a resource is static or dynamic is made based on the semantics of the mapping [8].

This definition of resource brings generality by encompassing many sources of information irrespective of its type or implementation. It allows dynamic binding of the reference to a representation and also allows the author to directly reference the concept rather than the representation of it [8].

Each resource is identified by an identifier which is assigned by the naming authority; they also perform the validation of semantics of mapping with the entity set [8].

2.6.2 Representation

The representation of each resource is a sequence of bytes along with its metadata that describes the sequence of bytes. The representation gives the present state of the resource[12]. The representation of a resource can be a document, image, file, instance, etc[8].

2.6.3 State Transfer

A web application which is designed RESTfully can be considered as a virtual state machine [11] as shown the Figure 2.1. The network of webpages of the application is seen as a state machine where each resource (Webpage) is shown as a node which is uniquely named by an URI. The user navigates through the application by selecting the links (state transition) which takes the application to the next state [11].

{Haupt, 2010 #27}

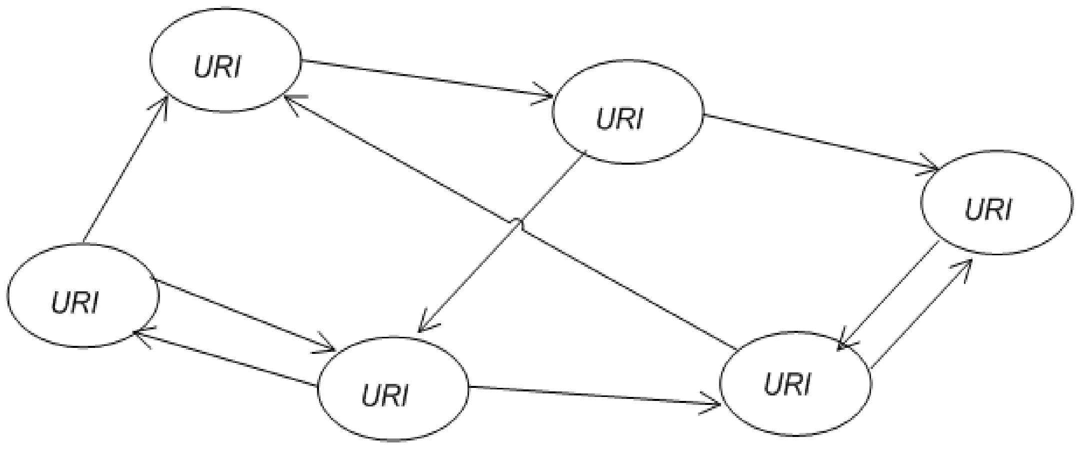


Figure 2.1 Web application as a finite state machine

As the name Representational State Transfer suggests, the clients and the resources transfer representations of states of the resource among themselves. In order to have meaningful communication between the client and the server, both of them should be in an agreement regarding the formation of the representations which are transferred. Most commonly used formats of communication are XML and JavaScript Object Notation (JSON). HTTP provides content negotiation among clients and servers which allows the clients to set the "Accept" header to specify the format which the client is willing to accept [13].

2.6.4 Design web application RESTfully.

The following are the steps performed while designing a web application using REST [11, 12].

1. Identify all conceptual entities to be exposed as resources.
2. Create URI for each resource. The resources should be nouns, not verbs.
3. Categorize your resources according to whether can just receive a representation of the resource ("GET"), or whether the client can modify (add to) the resource ("POST", "PUT", "DELETE").
4. All "GET" resource requests should be side-effect free.
5. Put hyperlinks within the resource representations to enable clients to drill down for more information, and/or to obtain related information.
6. Design to reveal data gradually.
7. Specify the format of response data.
8. Cache implementation to make the service efficient.
9. Implement stateless communication.

In REST, the client and the server are connected by a stateless HTTP connection. The advantage with a stateless connection is that the server treats each of the client's requests independently irrespective of the previous requests. So the server does not keep track of the clients' requests. If a client request fails to complete, then it cannot affect the subsequent requests as they are not dependent on the previous request. But the disadvantage of using a stateless connection is that it increases the repetitive data in a client request. As the server does not save any information regarding a previous request, the client should resend all the information needed for the server to process the request each time the client talks to the server [8, 14].

2.7 Literature review on REST

As discussed in section 2.5, Representational State Transfer (REST) is an architectural style which is relatively simplistic and less complex to use, as it provides a clear definition of its trade-offs and constraints [13]. Initially the concept of REST was designed only to support hypermedia document browsing, but these same principles are found to be very useful in developing complex distributed systems over the web [15].

The rich HTTP features enable the REST architecture style to be used in the development of high-level distributed systems. The RESTful architecture along with the XML-based information transfer paradigm has made it easy to aggregate various services which are called mashups. Typically it is not an easy task to create a mashup; it takes a lot of programming along with proper understanding about the APIs of all the services involved. In order to solve this problem, leading companies have developed tools that can mashup services which can be used even by a novice user. Some of such available tools are: Yahoo! Pipes, IBMs QEDWiki, and Google mashup Editor. These tools allow the user to specify a limited number of RESTful services and then connect these services one with another to form a mashup. But the main limitations of these tools are that they can aggregate a small number of services, and mostly these services are internal services of a company who developed the tool (e.g.: Google mashup tool can use Gmail, Google maps). These services should have only the standard output formats like RSS or ATOM. The services whose output is not a standard format cannot be mashed up using these tools which leaves out a large number of services. In order to address these limitations of scalability and complexity, Lathem J. et al. has proposed a framework called the SA-REST (Semantic Annotation-REST) which added semantics to the RESTful services. The SA-REST defines semantics for inputs, outputs, operations and faults of the RESTful services, which helps in easily aggregating the services to form Semantics Mashups (Smashups). As the RESTful services are annotated, the Smashups know everything about the inputs and the outputs, so that the data mediation can be automated [16].

Andreozzi S. et al., has proposes a RESTful way of developing a standard job submission and management interface for a Grid system. Their current specification for the services is based on the Web Service Description Language (WSDL) for creating, monitoring, maintaining and querying the services. They could successfully map the current Basic Execution Services (BES) specification to that of the HTTP actions to be performed on the resources. The authors also claim that the RESTful implementation is very easy and simple to implement when compared to the current specification [17], which suggests that the RESTful way of implementing MPR would be not only easy but also efficient when compared to developing it using web-services.

Yan L. et al. gave a process for reengineering an existing web application into a Restful web application. Most of the web applications which they considered are SOAP/RPC based [18]. They claim that it is beneficial to move from the traditional methods to the RESTful way of implementing a web application.

All the current research indicates that there is a wide range of benefits to developing a web application RESTfully like: ease of implementation, scalability, statelessness, caching, content-negotion, and most of all, all the operations on the resources can be carried out using the four HTTP methods (GET, PUT, POST and DELETE). But none of the current research addresses the implementation of a RESTful application that involves AJAX in it. AJAX makes an application look like a stand alone desktop application; the web pages are updated without refreshing the entire page, which means that the URI of the page remains the same but some part of the page is updated by using an AJAX call. In such a case, as we are using REST, there should be a way of identifying the webpage which has been updated by making AJAX calls because each webpage is a resource of the application. Our research concentrated on overcoming the complexities caused by the use of AJAX in a RESTfully designed application. If all the web pages, including the pages which were updated by making AJAX calls are named successfully then we believe that it is possible to bookmark any page and also implement the back button feature for such applications. As REST has been picking up its pace only very recently, we identified that these issues were not addressed in the literature so far, which gives us motivation and opportunity to solve these issues.

CHAPTER III

DESIGN USING REST

The software engineering process of designing a RESTful application is discussed in chapter 2, but the introduction of AJAX in a web application adds challenges in order to design it RESTfully using the same software engineering process. In an AJAX-based application, the webpage consists of containers, which display a representation of resources where as in a traditional webpage the entire webpage displays a representation of a resource, therefore it is challenging to handle several representations of resource in an AJAX page when compared to only one representation of one resource in a traditional webpage. Therefore, a new software engineering process of designing an AJAX-based web application using the Representational State Transfer architecture style is specified in section 3.1.

The proposed software engineering process is then implemented on the Material Properties Repository (MPR).

3.1 Software engineering process of designing an AJAX-based application RESTfully

Following is the software engineering process of designing an AJAX-based web application RESTfully

1. Any AJAX-based web application would have several containers in the page, which are updated without reloading the page. Identify all such containers.
2. Establish the relationship between the containers, i.e. how one container affects the other.
3. Define each of the states in the application.
4. Define the finite state machine for the application.
5. Define a URI schema for naming these states.
6. Name each of the states with the defined URI schema.
7. Identify all the onclick events that can cause the state transition of the application.
8. Save all the information related to the containers when an onclick is performed.
9. Create a URI by using above information.
10. Use the URI to identify and get the state.
11. Remember all the states of the FSM by remembering the URIs of the states.
12. Bookmarking a state now only means remembering the URI of the state.
13. Use the above remembered URIs to navigate to the back and forward states of the FSM

3.2 Generalizing the design

The generalized view of the RESTful design of the AJAX-based application can be seen in figure 3.1. This design can be used for any AJAX-based application to design it RESTfully.

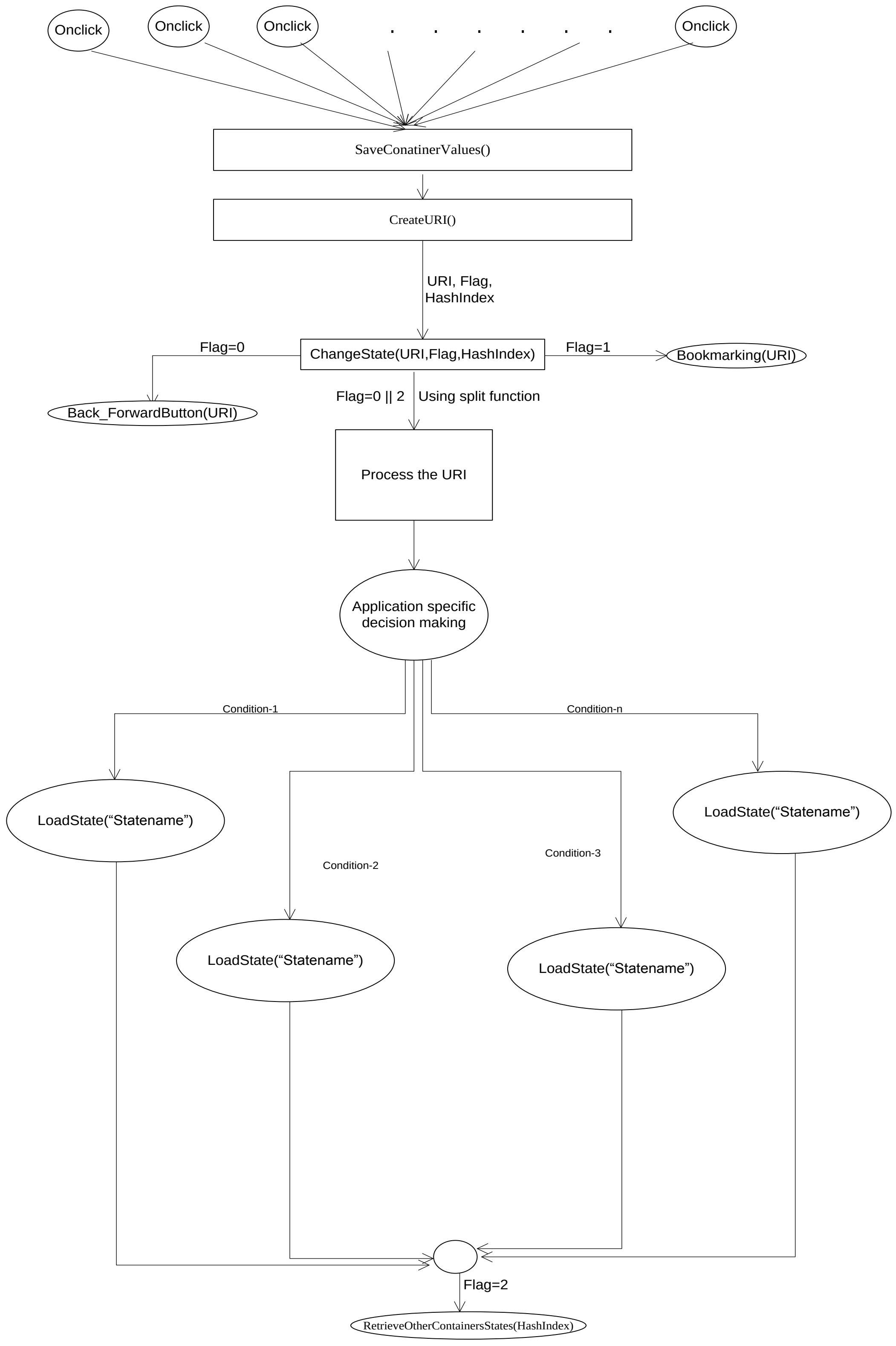


Figure 3.1 General design of an AJAX-based RESTful design

CHAPTER IV

CASE STUDY

4.1 Architecture of the MPR

The architecture of the Material Properties Repository can be viewed as shown in Figure 4.1. The user interface of MPR is designed using HTML along with JavaScript and AJAX. The requests and responses from the browser to the database pass through several layers, such as Tomcat, Service Bus, AXIS (WSDL and J2EE) and MATLAB. The backend database of MPR is mysql.

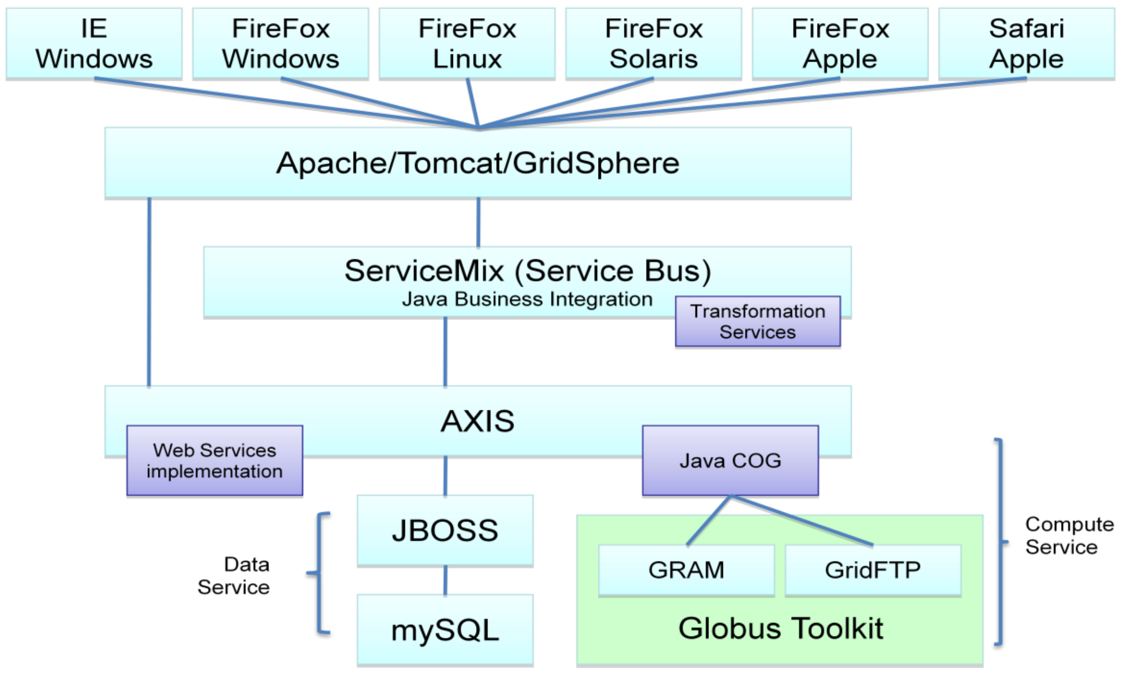


Figure 4.1 Architecture design of MPR

MPR is implemented on top of two web services: A) Data Service and B) Compute Service

A) Data Service:

Data Service aggregates three independent sub-services: metadata service, storage service, and replica locator.

Each experimental data set in the repository is stored in a file system. The storage service manages the part of the file system designated to store the data sets. When a file is submitted to the storage service, the service determines the location at which the file is to be stored, and returns its URI (Uniform Resource Identifier) to the caller. GridFTP is used as the transport mechanism for moving the files to and from the storage.

Metadata service collects the information about data sets maintained by the storage service. The information is comprised of the file identification (a name assigned by the user, project, material, etc.), the data provenance (owner, date submitted, etc.), tags enabling querying the metadata repository to find particular data sets matching search criteria, and some additional information necessary to process the data (such as transformation from raw force-displacement measurement to stress-strain relationship). When a new metadata record is created, the service returns its URI, so that it can be referred to at a later time. The metadata repository is implemented as a DBMS application.

The replica locator provides mappings between the metadata records and data files. Typically, entering a new data set to the MPR is performed in three steps. The metadata record is created ($URI_{metadata}$), the file is uploaded through storage services (URI_{data}), and finally, both URI are sent to the replica locator, where the mapping between them is established. Similarly, retrieving a data set from the MPR is done in three steps. Firstly, the user queries the metadata repository to retrieve $URI_{metadata}$ of the desired entry. Secondly, URI_{data} is retrived from the replica locator. Finally, the file is retrieved by submitting URI_{data} to the storage service. (Caching of URI_{data} does not work, as during each step authorization is made).

Because of this complexity, the Data Services is implemented as a façade. A facade is service that provides a simplified interface to a larger body of code, such as a class library, or a collection of independent services. The use of the facade pattern has several advantages:

- It makes easier to use and understand underlying services, since the facade has convenient methods for common tasks;
- It makes the code that uses the services more readable, for the same reason;
- It reduces the dependencies of outside code on the inner workings of the collection of services, since most code uses the facade, thus allowing more flexibility in developing the system;
- It wraps a complicated collection of APIs with a single well-designed API [1, 19].

B) Compute Service:

The model calibration tools are implemented as Matlab applications. In order to utilize them in the Web environment, a web service referred to as Compute Service has been developed. A pool of Matlab instances is running in the back-end. Similarly to Data Service, the complexity of the Compute Service is hidden by a façade [1].

4.2 Functionalities of MPR

The MPR provides the list of material properties which are displayed in the form or material tree or projects tree. The MPR currently supports five types of data types which are stress-strain, microstructure image, stress-life, damage model and microstructure model. The datasets can be viewed in terms of metadata or as a plot. The datasets can also be opened in the respective tool for future analysis. The MPR also provides features for the users to uploaded new datasets and also to download the existing ones.

4.3 Implementation of the web client

The web client of the MPR is developed using AJAX.

Features of the web client are:

1. Access data from the server
2. Selective updating of the required parts of the webpage (containers) as opposed to refreshing the entire page.
3. Mashing up disparate web services (compute service and data service)

The overall design of the Material Properties Repository web client can be viewed as a set of containers (placeholders) as shown in Figure 4.2. Here some of the containers are static, and others are dynamic. The content of the static placeholders remain the same always and do not change any state and they are represented in red color. The content of the dynamic placeholders change their state depending on the actions performed by the user, and these placeholders are represented in blue color.

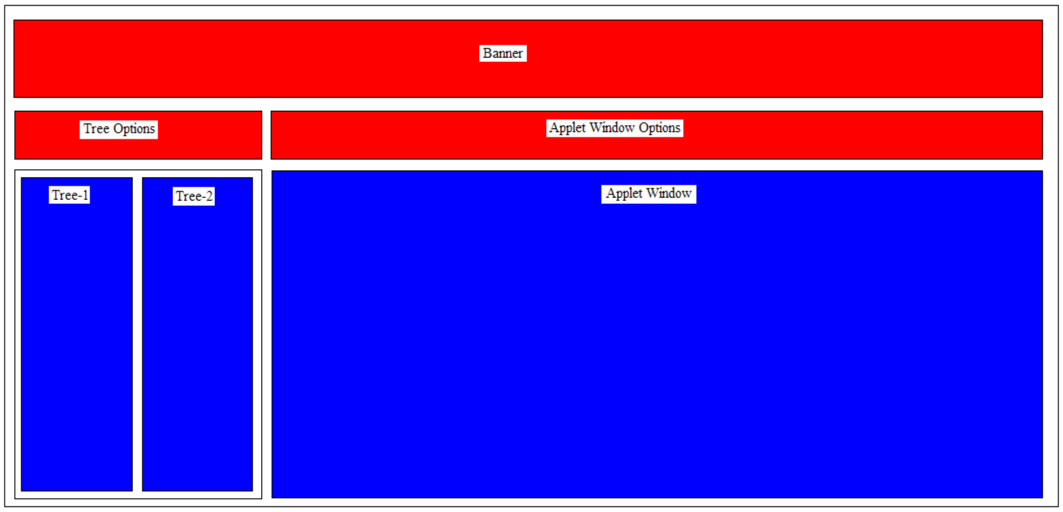
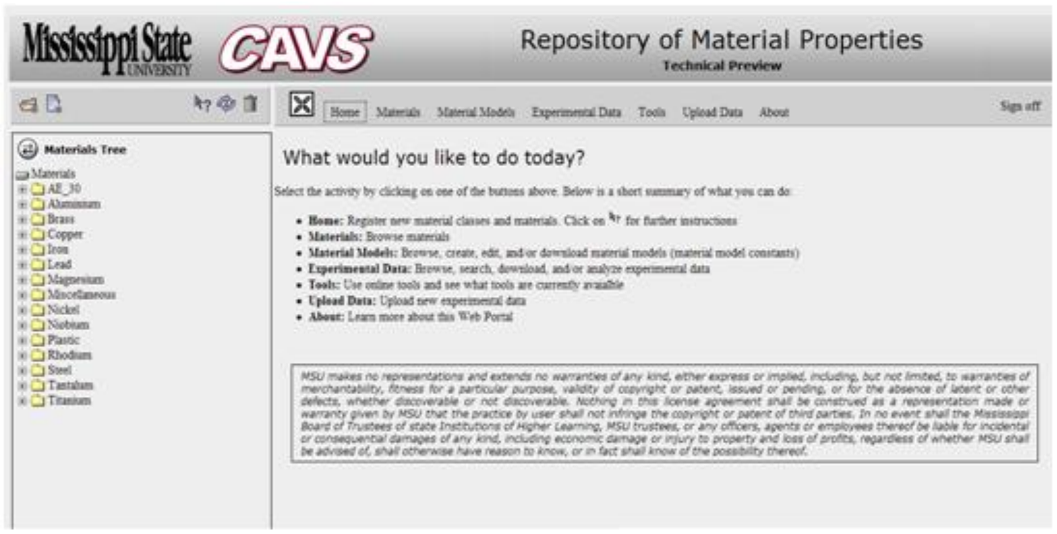


Figure 4.2 Layout of the MPR page

In order to update a container, the web client makes an AJAX call to change the content of the container without refreshing the complete page. Figure 4.3 shows the home page of the MPR page. When the user clicks on the ‘Materials’ tab of the ‘Applets window options’, then only the ‘Applet window’ is updated by making an AJAX call.

Figure 4.4 shows that the container (Applet Window), enclosured in red has been updated without reloading the entire page. All other containers remain the same.



Repository of Material Properties

Technical Preview

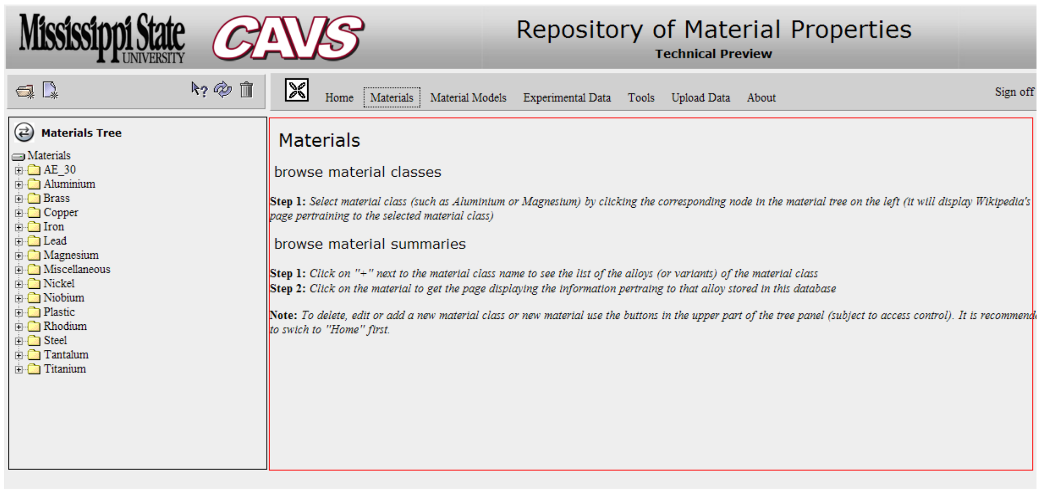
What would you like to do today?

Select the activity by clicking on one of the buttons above. Below is a short summary of what you can do:

- **Home:** Register new material classes and materials. Click on for further instructions
- **Materials:** Browse materials
- **Material Models:** Browse, create, edit, and/or download material models (material model constants)
- **Experimental Data:** Browse, search, download, and/or analyze experimental data
- **Tools:** Use online tools and see what tools are currently available
- **Upload Data:** Upload new experimental data
- **About:** Learn more about this Web Portal

MSU makes no representations and extends no warranties of any kind, either express or implied, including, but not limited, to warranties of merchantability, fitness for a particular purpose, validity of copyright or patent, issued or pending, or for the absence of latent or other defects, whether discoverable or not discoverable. Nothing in this license agreement shall be construed as a representation made or warranty given by MSU that the practice by user shall not infringe the copyright or patent of third parties. In no event shall the Mississippi Board of Trustees of state Institutions of Higher Learning, MSU trustees, or any officers, agents or employees thereof be liable for incidental or consequential damages of any kind, including economic damage or injury to property and loss of profits, regardless of whether MSU shall be advised of, shall otherwise have reason to know, or in fact shall know of the possibility thereof.

Figure 4.3 MPR home page screenshot



Repository of Material Properties

Technical Preview

Materials

browse material classes

Step 1: *Select material class (such as Aluminium or Magnesium) by clicking the corresponding node in the material tree on the left (it will display Wikipedia's page pertraining to the selected material class)*

browse material summaries

Step 1: *Click on "+" next to the material class name to see the list of the alloys (or variants) of the material class*
Step 2: *Click on the material to get the page displaying the information pertraing to that alloy stored in this database*

Note: *To delete, edit or add a new material class or new material use the buttons in the upper part of the tree panel (subject to access control). It is recommende to swich to "Home" first.*

Figure 4.4 Container updated using AJAX

4.4 Current state of the MPR and barriers for further development

The MPR is currently deployed as a “technical preview” at CAVS web site (ccg.hpc.msstate.edu/cmd) and made available to CAVS researchers and students as well as to a limited number of external users – the participants of the Integrated Computational Material Engineering (ICME) project performed by the U.S. Automotive Materials Partnership (USAMP). The feedback from our users is a very valuable source for making the assessment of the quality of the MPR design and its implementation. Not surprisingly, the users’ comments and requests for improvements, together with our experience with satisfying those requests revealed some flaws in the original design and implementation strategy. The flaws fall into two general categories: inefficiencies in the Graphical User Interface (GUI) and a poor maintainability of the system, in particular to the Web layer (shown as “Apache/Tomcat/GridSphere” box in the architectural diagram in Figure 4.1). Some of the problems identified are listed below.

4.4.1 Inefficiency in the Graphical User Interface (GUI)

There are four inefficiencies, which are as follows:

i. The implementation is inconsistent, similar functionalities are implemented multiple times, each time in a different way. As mentioned before, MPR uses AJAX to update the web page rather than reloading the entire page but the implementation of the tools are different from the implementation of the other features. The materials tree that is reloaded when a tool is opened is implemented differently using a different code. Each tool has its own upload feature integrated into it that is also provided in the MPR main page. The upload feature is available in two different places and both of the features are implemented differently, which are inconsistent with each other. Our aim is to make all the features and implementations work consistently.

ii. The GUI is inconsistent, difficult to maintain and extend (to add new functionality)

The GUI of the MPR has an inconsistent layout, the ‘Material Models’, ‘Experimental Data’, ‘Tools’ and the ‘Upload’ tabs have a description container inside the Applet window container which is not present in the layout of the other tabs.

iii. Navigation is difficult and confusing for the user

In order to get the metadata of a ‘Material model’, the user should go to the ‘Material Models’ tab, then click the ‘Browse by material’ button, then the data type drop down menu is displayed for the user to select the datatype. After selecting the data type the user should click on the material from the material tree, if the selected material has the material models of the selected data type then a second tree will be displayed that lists the material models. If the material models are available then the user should select a model and then click the ‘view metadata’ button that is available in the applet window in order to view the metadata of the material model. The sequence of steps to be followed in order to get what the user wants is very confusing in the current implementation. Similarly viewing the metadata or the plot of the experimental data is confusing. The navigation complexity should be reduced by following a consistent implementation.

iv. It is difficult for the user to repeat a previously performed analysis.

If a user is in the ‘Materials’ tab of the MPR page and then selects the ‘Materials Models’ tab, then the material models page is displayed. At this point, if the user clicks the back button, in the current implementation the user cannot go back to the ‘Materials’ tab, as the entire page is not refreshed, the browser assumes that we are on the same page. So the implementation should be improved such that the user can go to a previous page by clicking the back button.

One other problem with the current implementation is that it is not possible to save the webpage, i.e. bookmarking a result is not possible. When a user opens the metadata of a material, and then selects the bookmark option to save the page, the user cannot retrieve the metadata of the material which the user intended to save. Bookmarking is a feature which the users want in this application which is not possible in the current implementation. These problems can be solved by properly naming the web pages (resources).

4.4.2 Poor maintainability

The MPR was designed by several developers because of which different features are designed differently and there is redundancy of code. In order to modify a feature or add a new feature, it needs several changed to be made in several locations of the code.

4.5 Future improvements

4.5.1 Applying the formal software engineering methods

We plan to use the Iterative Development Model (IDM) to improve the design and implementation of the MPR. In the IDM, the development of the software is done in increments. In each increment there are three phases, which are Design, Implementation and Analysis as shown in Figure 4.5. New features are added to the software in iterations. The main advantages of this model are that the software is developed in increments in a systematic process rather than ad-hoc and it also reduces the amount of testing, as the testing is confined only to the newly added feature in each iteration.

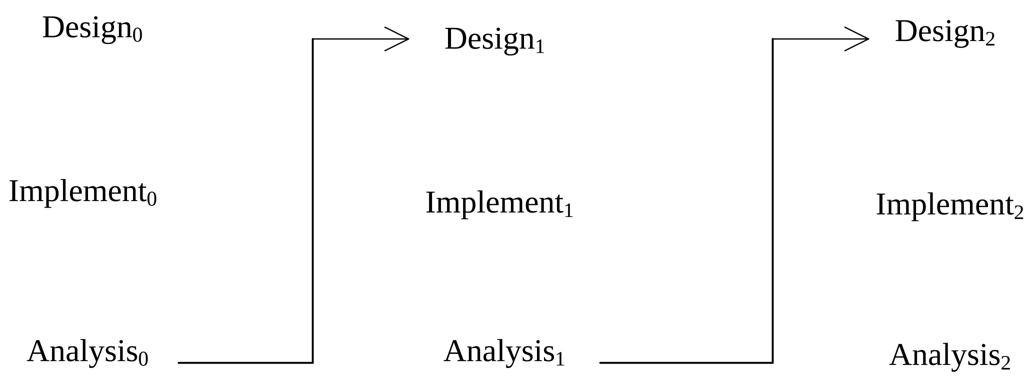


Figure 4.5 Iteration model

Each iteration consists of three phases-design phase, implementation phase and analysis phase. The iterations are done until all the items of the project control list are finished. The project control list has all the tasks to be performed to develop the final product.

This model can be used in this project as new requirements are continuously evolving. Whenever a new requirement is provided by the clients then these new requirements are implemented in the next iteration and only the part of the system that has been changed or updated is tested rather than the whole system.[20]

4.5.2 Applying the REST approach

That is, identify the resources and their representations to be displayed in the GUI, and define the user actions as the state transfers. This new design approach will replace the current design centered on the invoking methods on the remote (back-end) objects. The benefits of REST architectural style that will lead to both improvement of the GUI design and the maintainability of the code are discussed in Chapter II. The originality of our approach is that we are making an attempt to extend the REST approach to include AJAX-based web applications providing a Rich User Interface (RUI).

4.6 Overview of the design

The complete design of the MPR using the software engineering process from section 3.1 is followed as shown in the Appendix A. The overview of the RESTful design of the MPR can be put together as shown in figure 4.6, which include saving the information of each state, creating URIs, state transition, bookmarking and back-forward button implementation. A parameter called Flag is introduced in to this design to make decision regarding when to call the Bookmarking(), Back_ForwardButton() and RetrieveOtherContainersStates().

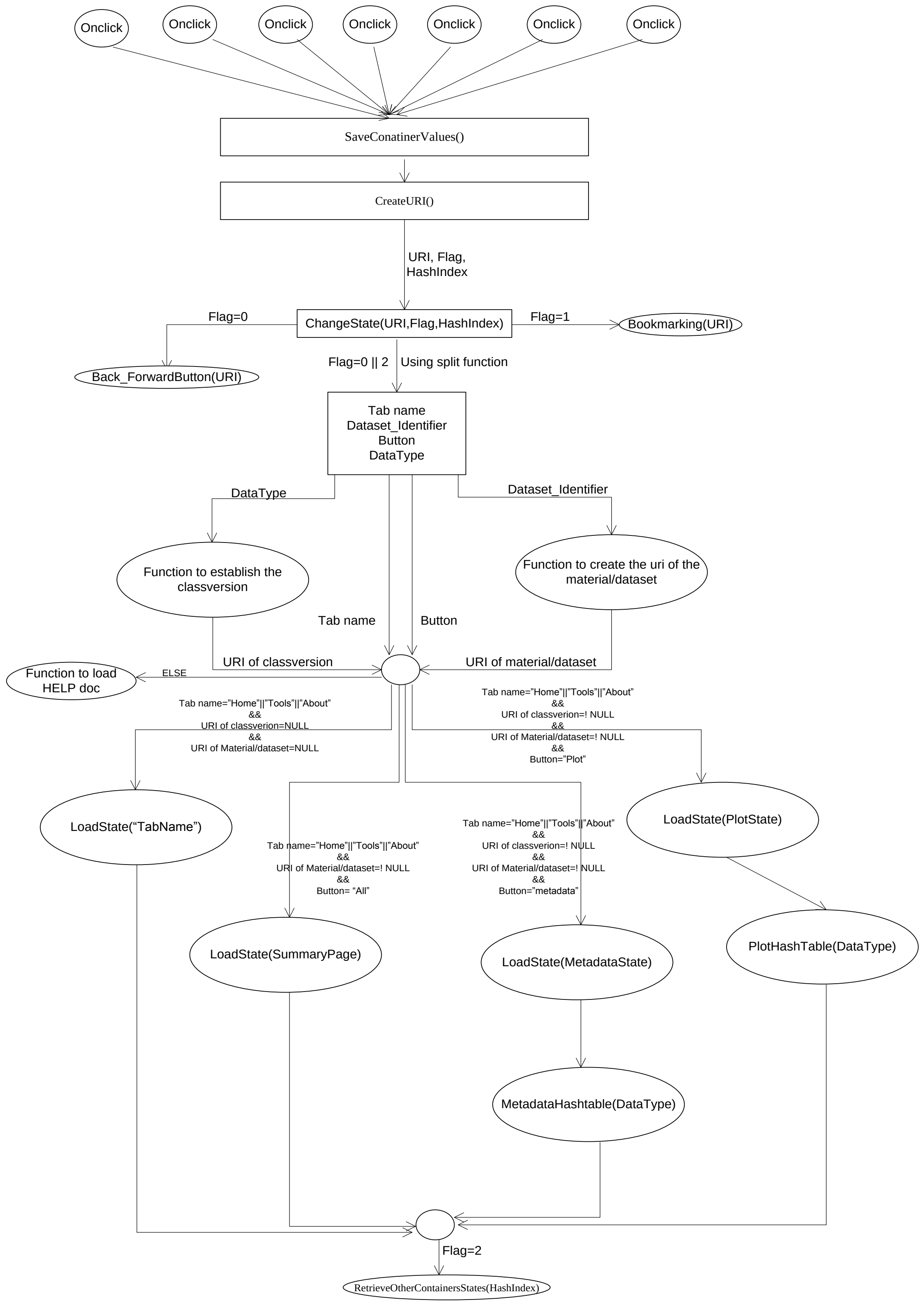

Onclick
Onclick
Onclick
Onclick
Onclick
Onclick
Onclick
SaveConatinerValues()
CreateURI()
URI, Flag, HashIndex
Flag=0
ChangeState(URI,Flag,HashIndex)
Flag=1
Bookmarking(URI)
Back_ForwardButton(URI)
Flag=0 || 2
Using split function
Tab name
Dataset_Identifier
Button
DataType
DataType
Dataset_Identifier
Function to establish the classversion
Function to create the uri of the material/dataset
Tab name
Button
URI of classversion
URI of material/dataset
Function to load HELP doc
ELSE
Tab name="Home"||"Tools"||"About" && URI of classverion=NULL && URI of Material/dataset=NULL
Tab name="Home"||"Tools"||"About" && URI of classverion=! NULL && URI of Material/dataset=! NULL && Button="Plot"
LoadState("TabName")
LoadState(PlotState)
Tab name="Home"||"Tools"||"About" && URI of Material/dataset=! NULL && Button= "All"
Tab name="Home"||"Tools"||"About" && URI of classverion=! NULL && URI of Material/dataset=! NULL && Button="metadata"
LoadState(SummaryPage)
LoadState(MetadataState)
PlotHashTable(DataType)
MetadataHashtable(DataType)
Flag=2
RetrieveOtherContainersStates(HashIndex)

Figure 4.6 Overall view of the design

The concept of the design is as follows:

1. Onlick events lead to saving all the containers information in a hashtable
2. Create the URI of the state of the MPR from the hashtable
3. After the URI is created call the ChangeState function by passing the URI, hashtable index and the Flag (If flag =0, then MPR is moving to a new state, if flag=1, Bookmark the page, if flag=2, retrieving a previous state or retrieving a bookmarked state)
4. URI is split to determine the Tab name, Classverion, URI of dataset and datatype.
5. Call the loadState which is a hashtable which intern uses LoadApplet for metadata and plot, MetadataHashTtable and PlotHashTable are used
6. When the flag=2, i.e. when a state is being retrieved, after loading the applet window with the correct information, all other containers are transferred to the correct state by calling the RetrieveotherContainersStates(HashIndex). This function reads the hashtable which stored all the information of all the other containers other than applet window and resets them.

4.7 Methodology for redesigning the MPR

As mentioned in section 4.4, the methodology used to redesign the MPR is Iterative Development Model (IDM).

The following are the improvements made in each iteration.

1. Remove redundant features and code.
2. Reduce the complexity – make metadata as the default action of leaf selection.
3. Reduce the number of inner containers in a container.

4. Add new features to the button bar like, download, upload, create folder and file.
5. New features for the project tree – upload, view and download files.
6. Improve the interface – banner image, color, less number of clicks to reach a state, more interactive, etc.
7. Add more information to Home, Tools and About tabs.
8. New help document.
9. Bookmark feature.
10. Back-Forward button feature.

CHAPTER V

RESULTS

5.1 Efficient design

The RESTful design can be considered to be an efficient design as it helped in identifying and removing the redundant features and redundant code.

- Redundant features: The RMI design has several featured, which performed similar operations. While defining states of the application these features were very easy identified and merged into one. Example: removal of experimental data and material model tabs
- Redundant code: The RMI design had several definitions of classversions and datatypes. But in the RESTful design each of these is a resource; therefore all these classversion and datatypes resources are defines at one location and are referenced by using the URIs. For example, there were several locations where classversions were defined in the previous design but in the RESTful design the classversion are defined only in one location in the ClassverionHashTable.

5.2 Easy to maintain

The RMI design everything is defined in terms of methods, where as in RESTful design everything is defined in terms of resources and these resources are addressed using URIs. In RMI design there are different methods for each event on the webpage, where as in RESTful design all the actions for any event in the webpage would results in getting the desired resource by passing the URI. This same action is performed to show any resource which make its very easy to maintain. If there is a new feature that needs to be added to the application then only the list of the resources is updated and the new resource is addressed by a URI.

5.3 Extensible

The RESTful design supports extensibility. As everything is defined in terms of a list of resources, if the application has to be extended then a new resource should be added to the list of resources. For example if the MPR should be extended with a new state, then that new state should be defined in the LoadState hash table. So the changed that should be made to extend the application is confined only toe defining the new resource.

5.4 Efficient GUI design

Reduced complexity of application: The RMI design had more states when compared to the REST design despite both performing the same features. The complexity can also be measured in terms of the number of clicks needed to reach a state. The RESTful design requires less number of clicks to reach a state when compared to RMI design as shown in table 5.1, and it also removed several internal containers present in a container.

Table 5.1 Complexity of RMI design vs. REST design

State	**RMI design** **(No. of clicks)**	**REST design** **(No. of clicks)**
Metadata	7	3
Summary page	3	2
Plot	8	4

CHAPTER VI

CONCLUSIONS

6.1 Contributions

The following are the contributions of my thesis:

1. The software engineering process of designing an AJAX-based web application RESTfully.
2. Designing an AJAX-based application RESTfully resolve redundancy in code and features, increases the maintainability of the code and supports extendibility.
3. A method of implementing the bookmark and back-forward button features for an AJAX-based application, which is otherwise not possible with web services.
4. A better implementation of the Material Properties Repository (MPR), which is developed at CAVS.

6.2 Future work

So far this document has explained the design of an AJAX-based web application using the software architecture style called the Representational State Transfer (REST). But there is a lot of scope for improving this design and perform several analyses on the results of the design. Some of the improvements that can be accomplished in the future are as follows:

1. This design has evolved in several iterations. In each iteration different range of problems were addressed. In spite of, increasing the maintainability of code, there is a lot of scope of reducing the complexity. One of such things is to find a better way of making decisions to call the correct loadState function rather than having several if-else statements.
2. In this design, I addressed only the states where the applet window is modified, so this design can be extended to all the intermediate states of the finite state machine so that it is possible to retrieve the intermediate states as well.
3. In this design, I have used hash tables to load states and applets, but it can be investigated to see if there is a better way of implementing this.
4. The RESTful design has clear advantages over the RMI design which can be seen by the reduced number of lines of code, identify and remove redundant code and redundant features, increase maintainability and extendibility by well organized code based on the states and their URIs and reduce complexity based on the reduced number of clicks to reach a state. But these results can be analyzed by performing empirical studies to compare the RESTful design with the RMI design. The empirical experiments can be performed on the REST design understandability, changeability, maintainability and extendibility when compared with the RMI design.

5. The RESTful design is believed to be complete, i.e. all the features present in the RMI design are also present in the REST design, but this was concluded by performing ad-hoc testing and by comparing the states with the RMI features. But it would be better to create a complete test suite for the RESTful design to identify any missing or incorrect features of the design by following a standard testing procedure.
6. The tools available in MPR should be made extendible; the RESTful design is confined only to the MPR extendibility but not to the tools. Therefore this software engineering process can be used to make the design of the tools RESTful, which is similar to the MPR design and make it extendable.

6.3 Summary

This document discusses the software engineering process of designing an AJAX-based application using the software architecture style called the Representational State Transfer (REST). The discussion from the previous sections can be summarized say that the by incorporating the REST methodologies into the design of an AJAX-based web application that aggregates (mashes up) disparate services will lead to an efficient, easy to maintain, and extensible implementation and efficient GUI design which is in support of our initial hypothesis.

REFERENCES

[1] T. Haupt, "Cyberinfrastructure for the Integrated Computational Material Engineering", 2010 TMS Annual Meeting, Seattle, WA, February 15-19, 2010.

[2] S. Murugesan, "Understanding Web 2.0," *IT Professional,* vol. 9, no. 4, 2007, pp. 34-41.

[3] T. Fischer, et al., "An Overview of Current Approaches to Mashup Generation," *Proceedings: International Workshop on Knowledge Services and Mashups (KSM09),* 2009

[4] L. Paulson, "Building rich web applications with Ajax," *Computer,* vol. 38, no. 10, 2005, pp. 14-17.

[5] E. Galyon, "The advantages and limitations of using ajax for distributed application development," *Academic Computing and Networking Services,* 2006.

[6] W.W.W. Consortium, "Reconciling Web Services and REST Services," 2005; http://www.w3.org/2005/Talks/1115-hh-k-ecows/#(10).

[7] Wikipedia, "Representational state transfer," April, 17th, 2009
2009; http://en.wikipedia.org/wiki/Representational_State_Transfer.

[8] R.T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Doctoral Dissertation, University of California, Irvine, CA, 2000.

[9] A. Asaravala, "Giving SOAP a REST," 2002; http://www.devx.com/DevX/Article/8155.

[10] P. Freitag, "REST vs SOAP Web Services," 2005; http://www.petefreitag.com/item/431.cfm.

[11] R.L. Costello, "Building Web Services the REST Way," 2009; http://www.xfront.com/REST-Web-Services.html.

[12] Z. Jin, et al., “On Web Service Construction Based on REpresentation State Transfer,” *Proceedings: IEEE International Conference on e-Business Engineering,(ICEBE '08)*, 2008, pp. 665-668.

[13] S. Vinoski, “RESTful Web Services Development Checklist,” *IEEE Internet Computing*, vol. 12, no. 6, 2008, pp. 96-95.

[14] Wikipedia, “Stateless server,” 2009; http://en.wikipedia.org/wiki/Stateless_server.

[15] D. Roure, “Towards Computational Abstractions over a RESTful Architecture,” 2008, pp. 719-722.

[16] J. Lathem, et al., “SA-REST and (S) mashups: Adding Semantics to RESTful Services,” 2007, pp. 469-476.

[17] S. Andreozzi and M. Marzolla, “A RESTful Approach to the OGSA Basic Execution Service Specification,” *Proceedings: Fourth International Conference on Internet and Web Applications and Services, (ICIW '09)*, 2009, pp. 131 - 136

[18] L. Yan, et al., “Reengineering Legacy Systems with RESTful Web Service,” *Proceedings: 32nd Annual IEEE International conference of the Computer Software and Applications, 2008 (COMPSAC '08)*, 2008, pp. 785-790.

[19] V.K. Patil, “Facade Design Pattern”, http://aspalliance.com/970.

[20] P. Jalote, *An Integrated Approach to Software Engineering*, Narosa Publications, 2005.

APPENDIX A

DESIGNING THE MPR RESTFULLY

This appendix presents the RESTful design of the Material Properties Repository by following the software engineering process of designing an AJAX-based application given in section 3.1.

A.1 Identify all the containers present in the MPR webpage

Figure 4.2 shows the layout of the MPR with all the containers present. The current implementation of the MPR had inconsistent sub-containers in the applet window. Some of applets have sub-containers and some did not. This inconsistency has to be fixed so that they can support the RESTful design. Keeping these problems in mind the layout of the MPR has now been modified as shown in figure A.1 from the initial layout shown in figure 4.2.

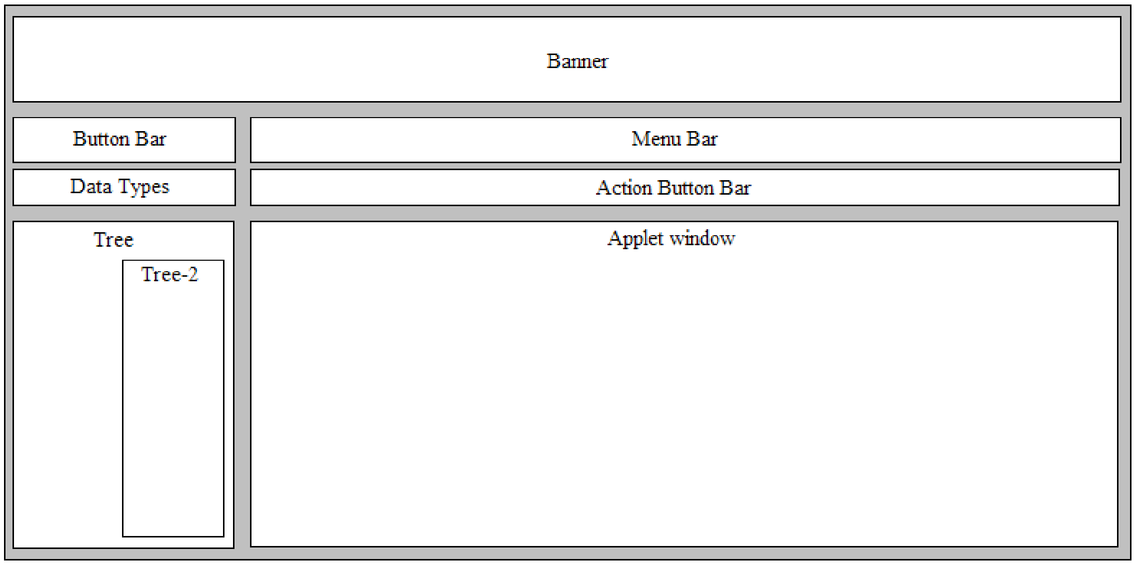


Figure A.1 Layout of the new design of the MPR

The layout of the MPR has been updated with two new containers, the data type container and the action button bar. These changes reduce the number of selections that should be made inside the applet window.

The following is the list of all the containers present in the MPR:

1. Banner
2. Button bar
3. Menu bar
4. Data Types
5. Action button bar
6. Tree
7. Tree-2
8. Applet window

A.2 Establish the relationship between the containers

The MPR webpage is comprised of containers as shown in figure A.1, which are the target for the AJAX responses. The “Banner” is the static part of the page, which contains the logo and the title of the application. The “menu bar” contains the home, tools, help and about tabs. The “button bar” contains new material class, new material, edit dataset or folder, refresh, download, upload and delete buttons. The “data type” container has a dropdown menu with the options all data types, stress-strain, microstructure image, strain-life, damage model and microstructure model. The “Tree” container has the materials tree as a default and a swap button, which can be used to display the projects tree. The “tree” container has a sub container called the “Tree-2”, when a data type value other than “all datatypes” is selected from the data types drop down menu and a material is selected from the materials tree then the “Tree-2” is displayed with the list of the datasets present in the material which are of the selected data type. The “Action button bar” has the action buttons, which are the metadata, plot, tool and summary page. The “Applet window” container has the home tab information as default, and it can display the content, which is determined by one or a combination of selections made in the menu bar, data types, tree and action button bar. The possible information that can be shown in the “Applet window” are home tab, tools tab, about tab, summary page, metadata or plot.

A.3 Define each of the states in the application

Definitions:

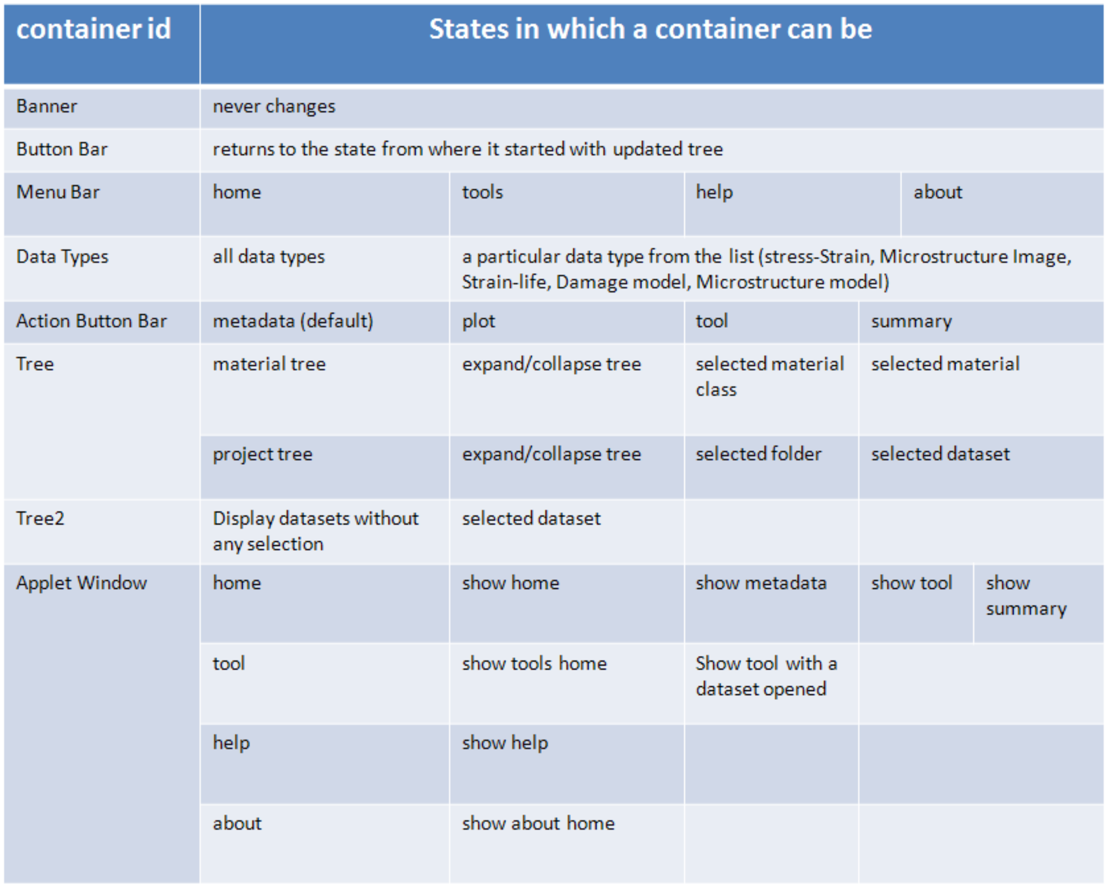
Resource: The intended conceptual target of a hypertext reference.

Representation: specific representation of a specific resource (like HTML document, JPEG image).

State: Representations of all the containers in a webpage at a given point of time.

Table A.1 shows the list of all the containers and their possible state. The state of individual containers is the representation of the resource, whereas the state of the webpage is the collective states of the individual containers. From the table, it can be observed that some containers are static, as they never change state (like banner and button bar), while other change states. This table helps us in defining the state of the webpage, which is an aggregate of the individual containers.

Table A.1 List of containers and their possible states



container id	States in which a container can be				
Banner	never changes				
Button Bar	returns to the state from where it started with updated tree				
Menu Bar	home	tools	help	about	
Data Types	all data types	a particular data type from the list (stress-Strain, Microstructure Image, Strain-life, Damage model, Microstructure model)			
Action Button Bar	metadata (default)	plot	tool	summary	
Tree	material tree	expand/collapse tree	selected material class	selected material	
	project tree	expand/collapse tree	selected folder	selected dataset	
Tree2	Display datasets without any selection	selected dataset			
Applet Window	home	show home	show metadata	show tool	show summary
	tool	show tools home	Show tool with a dataset opened		
	help	show help			
	about	show about home			

Each of these containers displays a representation of a resource, and the union of the container states is the page state. Any change to one of these containers would result in a change of state of the webpage. The number of states that can be achieved by different combination of these containers is very large, and some of them are not meaningful from the application point of view. Hence, we define states with respect to the applet window, each of the states have a different representation of the resource.

The following are the states that are selected for naming with a URI:

1) Home tab
2) Tools tab
3) About tab
4) Summary page
5) Metadata
6) Plot

A.4 Define the finite state machine for the application

In the MPR, each state can be achieved by different combinations of the selections made by the user in the “menu bar”, “Action button bar”, “Data types” and the “Tree”.

The following are the states of the MPR which include the states mentioned in section A.3 and the intermediate state while reaching the above 6 states:

1. Default home page: This is the initial state of the MPR with all the containers in their default states.
2. Home tab: Home tab information is displayed in the applet window with the home tab highlighted in the menu bar, while all other containers are in default states.
3. Tools tab: Tools information is displayed in the Applet window with the tools tab highlighted in the menu bar, while all other containers are in default states.

4. About tab: About information is displayed in the Applet window with the about tab highlighted in the menu bar, while all other containers are in default states.
5. Data type selected state: The drop down menu in the data types' container is set to any option other than all datatypes value. All other containers are in the default state.
6. Material Folder: A selected material folder is highlighted in the materials tree and all other containers are in default states.
7. Material folder with material classes: Material folder expands to display the material classes which are present in it while all other containers are in default states.
8. Tree-2 state: The tree-2 state has the data type menu set to a data type value different from all data types, the tree container has the materials tree with one of the material folders expanded and a material selected. The tree-2 container displays the list of the datasets in the selected material with the datatype specified in the data type drop down menu. All other containers are in default states
9. Projects tree: The projects tree is displayed in the tree container while all other containers are in default states.
10. Projects folder with projects and datasets: The tree container has the projects folder expanded to display the sub projects and the dataset present in it, and all other containers are in default states.

11. Summary page: Applet window displaying the summary of the material and the material name is highlighted in the materials tree with the material folder expanded. All other containers are in default states.
12. Metadata state: Applet window displays the metadata of a dataset and the dataset name is highlighted in the materials tree or in tree-2 or in the project tree. The tree container can be in either of the following three states:
    - The material folder expanded and a material is selected when the data type menu is set to all data types.
    - The material folder expanded and a material is selected and a dataset is selected from the tree-2 while the data type menu is set to any other option other than all data types.
    - The dataset is selected from the projects tree with a project folder opened while the data types menu can be set to any option.

    All other containers are in default states.
13. Plot state: Applet window displaying the plot of the selected dataset and the dataset name is highlighted in the materials tree or in tree-2 or in the project tree. The tree container can be in either of the following three states:
    - The material folder expanded and a material is selected when the data type menu is set to all data types.
    - The material folder expanded and a material is selected and a dataset is selected from the tree-2 while the data type menu is set to any other option other than all data types.

- The dataset is selected from the projects tree with a project folder opened while the data types menu can be set to any option.

All other containers are in default states.

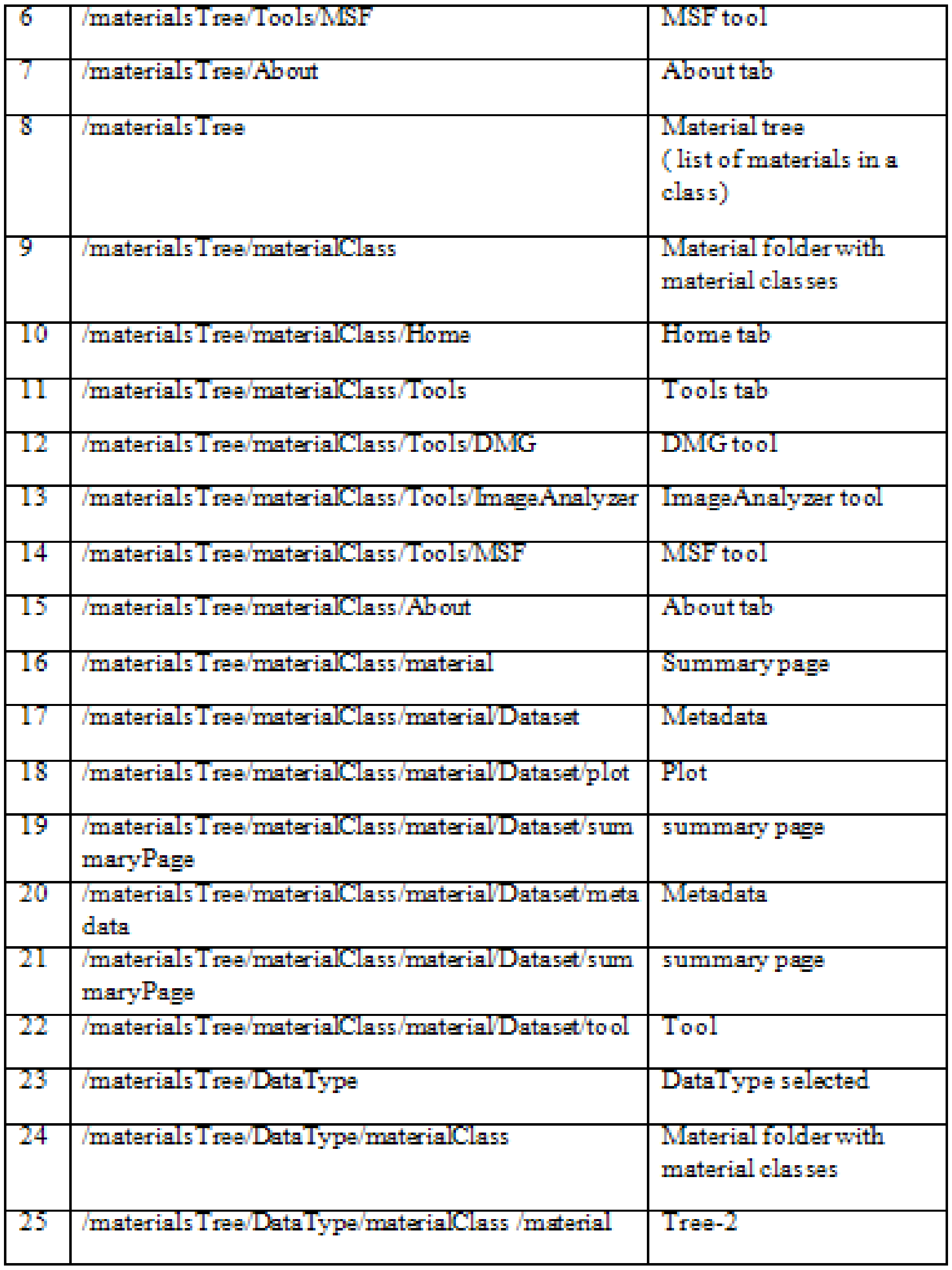
14. DMG tool state: The home page of the DMG tool is displayed, or the DMG tool with a dataset is displayed. This state requires the entire webpage to be reloaded.
15. ImageAnalyzer tool state: The home page of the ImageAnalyzer tool is displayed, or the ImageAnalyzer tool with a dataset is displayed. This state requires the entire webpage to be reloaded.
16. MSF tools: The home page of the MSF tool is displayed, or the MSF tool with a dataset is displayed. This state requires the entire webpage to be reloaded.

Table A.2 shows various combinations of these selections made from the default home page and the corresponding state that was achieved. From the table, we observe that a state is achieved in multiple ways.

Table A.2 List of paths and the corresponding states

No.	Path from default home page to other states	State achieved
1	/	Default home page
2	/materialsTree/Home	Home tab
3	/materialsTree/Tools	Tools tab
4	/materialsTree/Tools/DMG	DMG tool
5	/materialsTree/Tools/ImageAnalyzer	ImageAnalyzer tool

Table A.2 List of paths and the corresponding states (continued)



6	/materialsTree/Tools/MSF	MSF tool
7	/materialsTree/About	About tab
8	/materialsTree	Material tree (list of materials in a class)
9	/materialsTree/materialClass	Material folder with material classes
10	/materialsTree/materialClass/Home	Home tab
11	/materialsTree/materialClass/Tools	Tools tab
12	/materialsTree/materialClass/Tools/DMG	DMG tool
13	/materialsTree/materialClass/Tools/ImageAnalyzer	ImageAnalyzer tool
14	/materialsTree/materialClass/Tools/MSF	MSF tool
15	/materialsTree/materialClass/About	About tab
16	/materialsTree/materialClass/material	Summary page
17	/materialsTree/materialClass/material/Dataset	Metadata
18	/materialsTree/materialClass/material/Dataset/plot	Plot
19	/materialsTree/materialClass/material/Dataset/summaryPage	summary page
20	/materialsTree/materialClass/material/Dataset/metadata	Metadata
21	/materialsTree/materialClass/material/Dataset/summaryPage	summary page
22	/materialsTree/materialClass/material/Dataset/tool	Tool
23	/materialsTree/DataType	DataType selected
24	/materialsTree/DataType/materialClass	Material folder with material classes
25	/materialsTree/DataType/materialClass /material	Tree-2

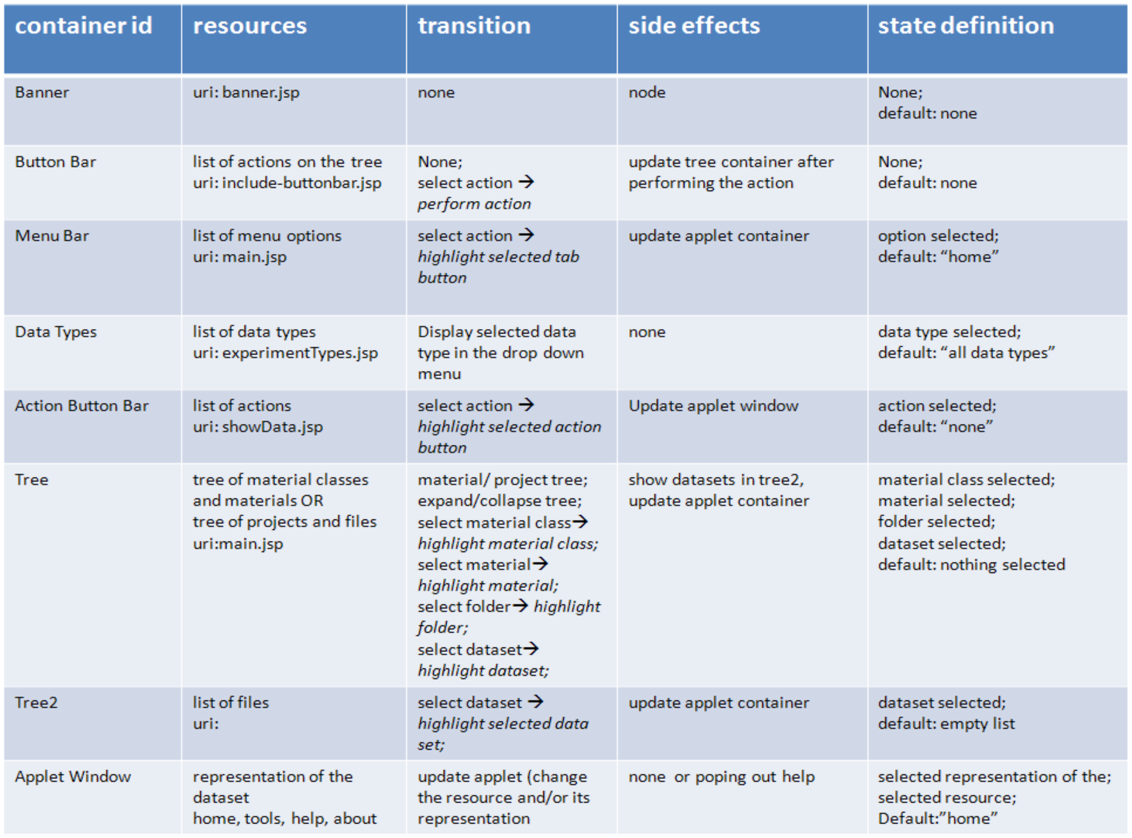
Table A.2 List of paths and the corresponding states (continued)

26	/materialsTree/DataType/materialClass /material/dataset	Metadata
27	/materialsTree/DataType/materialClass /material/dataset/plot	plot
28	/materialsTree/DataType/materialClass /material /dataset/metadata	Metadata
29	/materialsTree/DataType/materialClass/material/dataset/summaryPage	summary page
30	/materialsTreeDataType/materialClass/material /dataset/tool	Tool
31	/projectTree	Project tree (Display folders and datasets)
32	/projectTree/FolderPath	Project folder with sub projects and datasets
33	/projectTree/FolderPath/Dataset	Metadata
34	/projectTree/FolderPath/Dataset/plot	Plot
35	/projectTree/FolderPath/Dataset/metadata	Metadata
36	/projectTree/FolderPath/Dataset/tool	Tool
37	/projectTree/FolderPath/Home	home tab
38	/projectTree/FolderPath/Tools	tools tab
39	/projectTree/FolderPath/Tools/DMG	DMG tool
40	/projectTree/FolderPath/Tools/ImageAnalyzer	ImageAnalyzer tool
41	/projectTree/FolderPath/Tools/MSF	MSF tool
42	/projectTree/FolderPath/About	About tab

Figure A.2 shows the MPR as a finite state machine. The states that are shaded in grey are the ones which we have considered as resources and are named with URI schema, which is defined in section A.5. The states DMG tool, ImageAnalyzer tool and

MSF tool do not involve AJAX calls and require reloading of the entire page, and the remaining states are intermediate states. The numbers on the arrows indicate the path as defined in table A.2. Table A.3 shows the state transitions of each container and their side effects, which in turn are responsible for the state transition of the entire MPR webpage, as shown in Figure A.2.

Table A.3 List of containers and their details



container id	resources	transition	side effects	state definition
Banner	uri: banner.jsp	none	node	None; default: none
Button Bar	list of actions on the tree uri: include-buttonbar.jsp	None; select action → *perform action*	update tree container after performing the action	None; default: none
Menu Bar	list of menu options uri: main.jsp	select action → *highlight selected tab button*	update applet container	option selected; default: "home"
Data Types	list of data types uri: experimentTypes.jsp	Display selected data type in the drop down menu	none	data type selected; default: "all data types"
Action Button Bar	list of actions uri: showData.jsp	select action → *highlight selected action button*	Update applet window	action selected; default: "none"

Tree	tree of material classes and materials OR
	tree of projects and files
	uri:main.jsp
		material/ project tree;
		expand/collapse tree;
		select material class→ *highlight material class;*
		select material→ *highlight material;*
		select folder→ *highlight folder;*
		select dataset→ *highlight dataset;*
			show datasets in tree2,
			update applet container
				material class selected;
				material selected;
				folder selected;
				dataset selected;
				default: nothing selected

Tree2	list of files uri:	select dataset → *highlight selected data set;*	update applet container	dataset selected; default: empty list
Applet Window	representation of the dataset home, tools, help, about	update applet (change the resource and/or its representation	none or poping out help	selected representation of the; selected resource; Default:"home"

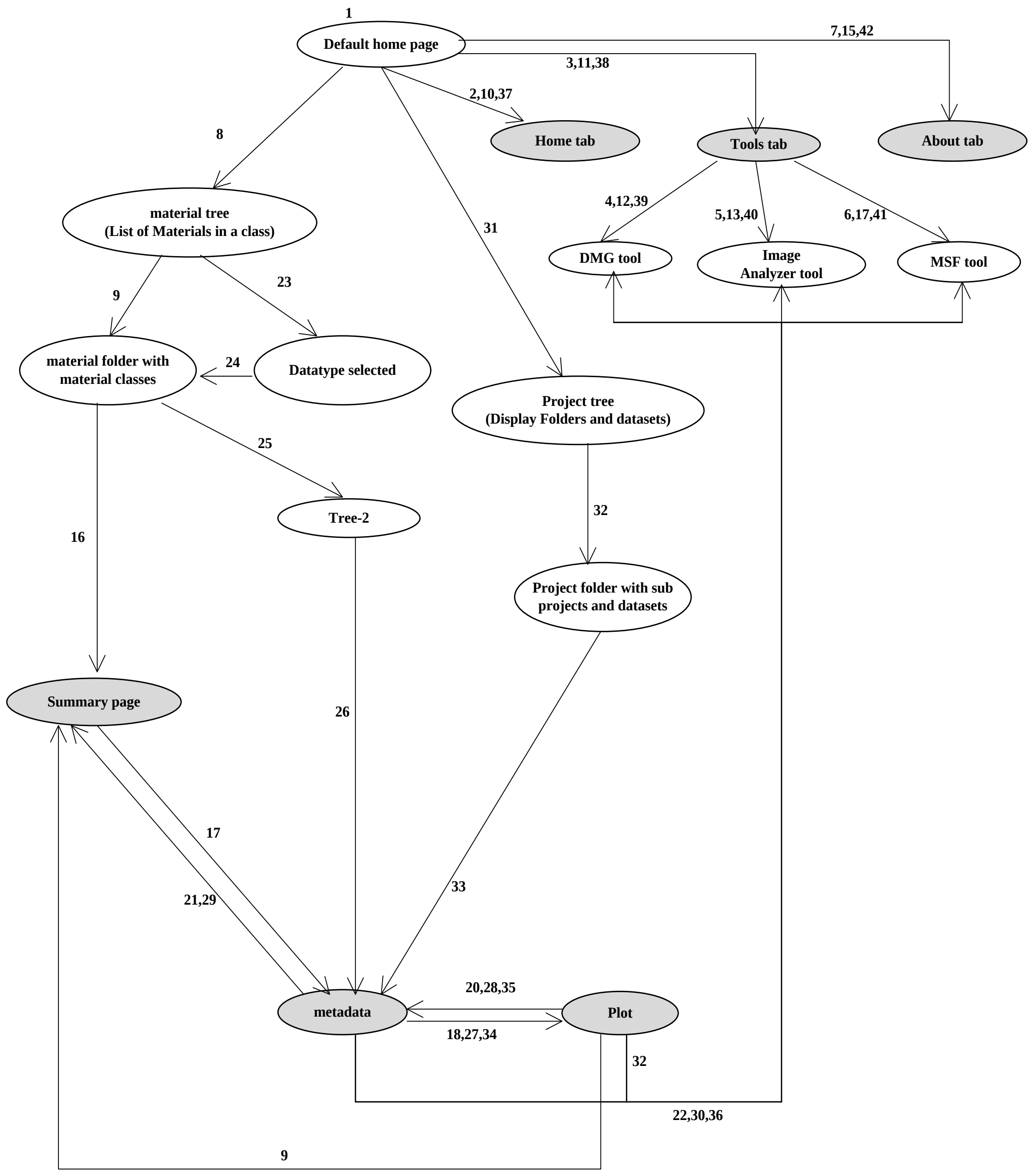
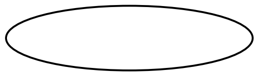


Figure A.2 MPR as a finite state machine

Diagram key:



Intermediate State or a state that requires reloading the page

⬭ States selected for naming with the URI

⟶ State transition

Numbers: The numbers on the arrow correspond to the path specified in table A.2.

Therefore, it is seen that the MPR can now be viewed as a finite state machine.

A.5 Defining a URI schema for naming the states

As discussed in previous sections, the state of the MPR is the aggregate of the states of the individual containers. Therefore, in principle, the URI of the webpage should contain a field for each of the containers. From table A.1 we can see that the values for the banner and button bar are none as they cannot go to any other state other than the default state. The content displayed in the applet window is determined by one selection or a combination of selections made in the menu bar, data types, tree (&tree-2), and action button bar, while the other containers, like banner and the button bar, do not affect the contents of the applet window. Therefore, the URI of the resource should contain information related to each of the four containers that affect the content of the applet window. The containers that affect the applet window are the menu bar, tree, action button bar, and the data types; therefore, there should be a reference related to these containers in the URI. Hence we choose the URI schema shown in figure A.3.

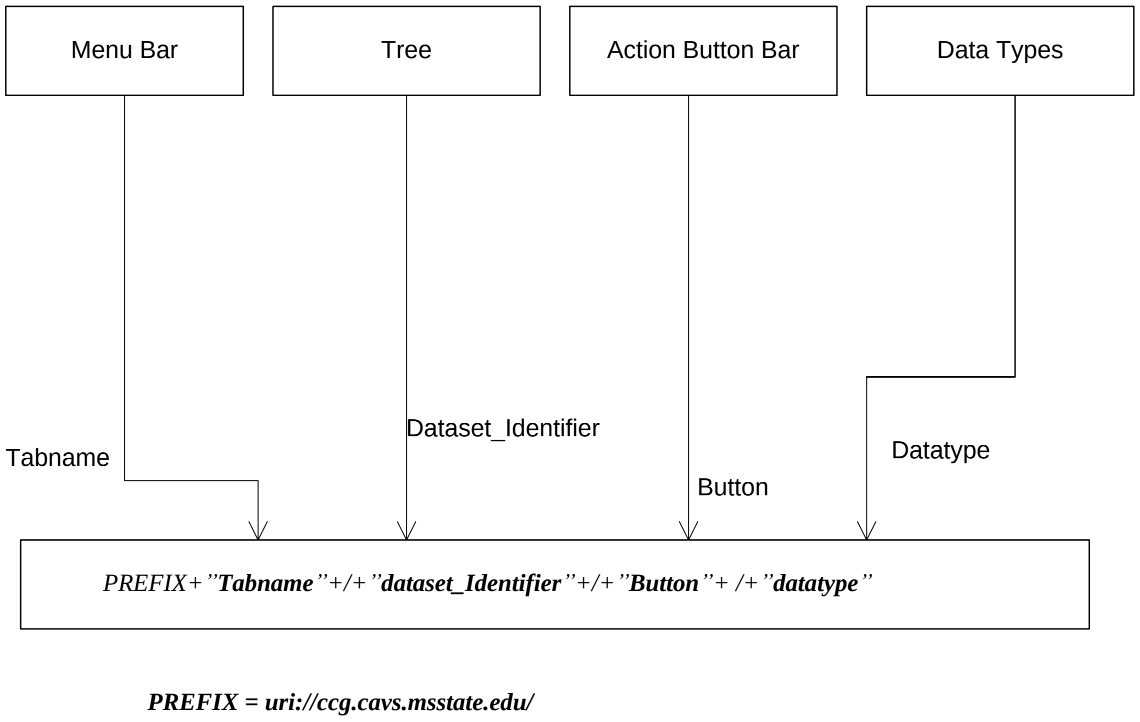

Menu Bar
Tree
Action Button Bar
Data Types
Tabname
Dataset_Identifier
Button
Datatype
PREFIX+"Tabname"+/+"dataset_Identifier"+/+"Button"+ /+"datatype"
PREFIX = uri://ccg.cavs.msstate.edu/

Figure A.3 URI schema

URI schema:

uri://ccg.cavs.msstate.edu/Tabname/dataset_Identifier/Button /datatype

The following are the possible values of each of the individual fields of the URI, provided "uri://ccg.cavs.msstate.edu" is the prefix which is present in all the URIs.

Tabname: Home (default), Tools, Help, About

***Dataset_Identifier*:** If the URI of the Dataset T1R4N4_A1_DMG.data is uri://ccg.cavs.msstate.edu/MaterialDB/MaterialDB/StressStrain.P.1/4798, then the value of the dataset_Identifier will be "4798."

Null is the default value.

Button: Metadata, plot, tool, Null is the default value.

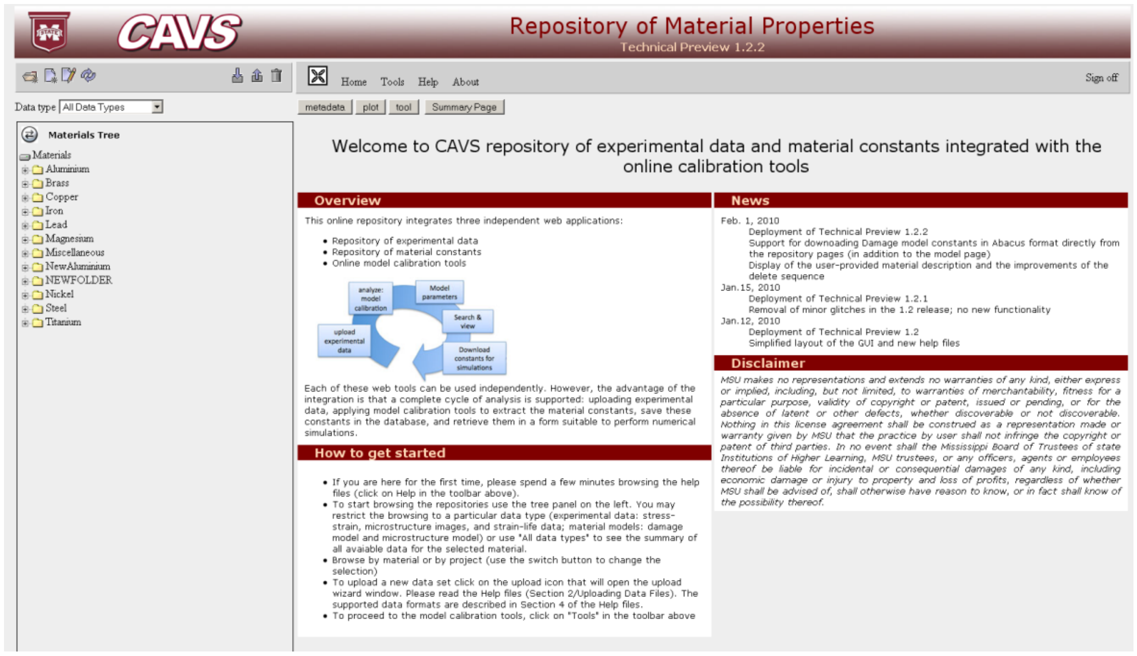
Datatype: All, StressStrain.P.1, MaterialImage, Fatigue.1, Microstructure, Damage, MultiStageFatigue and Null is the default value.

A.6 Name each of the states with the defined URI schema

We have defined 6 states in section A.3, and we name these states by using the URI schema as defined in section A.5.

The following is the list of each state and the URI with which it is addressed.

Home state:



Welcome to CAVS repository of experimental data and material constants integrated with the online calibration tools

Overview

This online repository integrates three independent web applications:

- Repository of experimental data
- Repository of material constants
- Online model calibration tools

Each of these web tools can be used independently. However, the advantage of the integration is that a complete cycle of analysis is supported: uploading experimental data, applying model calibration tools to extract the material constants, save these constants in the database, and retrieve them in a form suitable to perform numerical simulations.

How to get started

- If you are here for the first time, please spend a few minutes browsing the help files (click on Help in the toolbar above).
- To start browsing the repositories use the tree panel on the left. You may restrict the browsing to a particular data type (experimental data: stress-strain, microstructure images, and strain-life data; material models: damage model and microstructure model) or use "All data types" to see the summary of all avaiable data for the selected material.
- Browse by material or by project (use the switch button to change the selection)
- To upload a new data set click on the upload icon that will open the upload wizard window. Please read the Help files (Section 2/Uploading Data Files). The supported data formats are described in Section 4 of the Help files.
- To proceed to the model calibration tools, click on "Tools" in the toolbar above

News

Feb. 1, 2010
Deployment of Technical Preview 1.2.2
Support for downoading Damage model constants in Abacus format directly from the repository pages (in addition to the model page)
Display of the user-provided material description and the improvements of the delete sequence
Jan.15, 2010
Deployment of Technical Preview 1.2.1
Removal of minor glitches in the 1.2 release; no new functionality
Jan.12, 2010
Deployment of Technical Preview 1.2
Simplified layout of the GUI and new help files

Disclaimer

MSU makes no representations and extends no warranties of any kind, either express or implied, including, but not limited, to warranties of merchantability, fitness for a particular purpose, validity of copyright or patent, issued or pending, or for the absence of latent or other defects, whether discoverable or not discoverable. Nothing in this license agreement shall be construed as a representation made or warranty given by MSU that the practice by user shall not infringe the copyright or patent of third parties. In no event shall the Mississippi Board of Trustees of state Institutions of Higher Learning, MSU trustees, or any officers, agents or employees thereof be liable for incidental or consequential damages of any kind, including economic damage or injury to property and loss of profits, regardless of whether MSU shall be advised of, shall otherwise have reason to know, or in fact shall know of the possibility thereof.

Figure A.4 Screenshot of home tab state

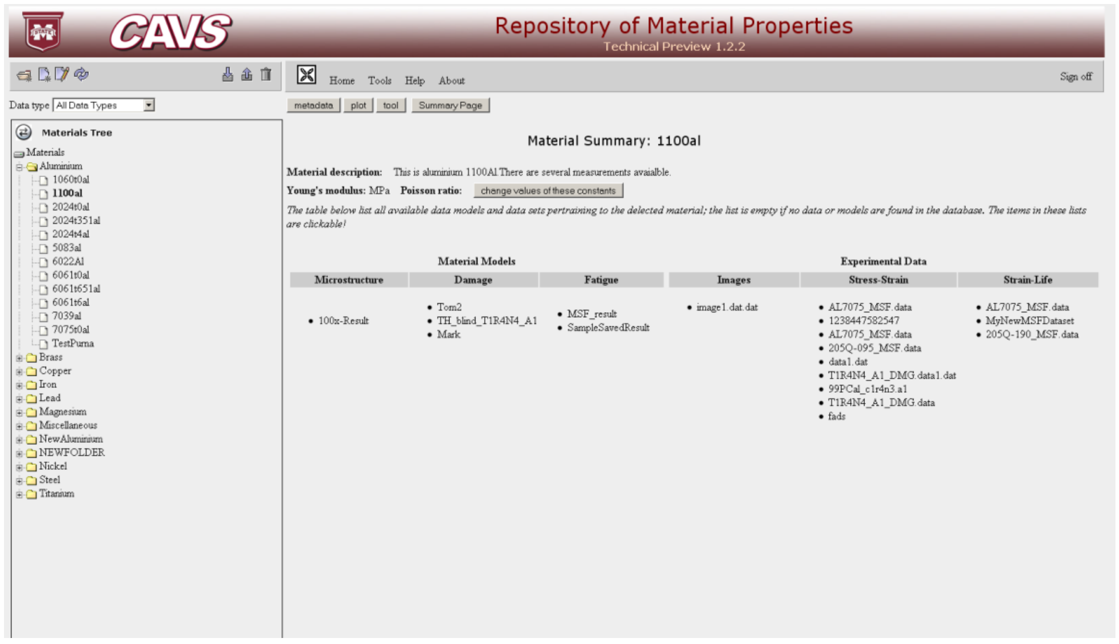
URI for home state:

uri://ccg.cavs.msstate.edu/Home

Table A.4 List of containers and their values in home state

Container name	Banner	Button bar	Menu bar	Data Types	Action Button bar	Tree	Tree-2	Applet window
Value	Default	Default	Home	Any	Default	Any	Any	Show Home

Summary page state:



Material Summary: 1100al

Material description: This is aluminium 1100Al.There are several measurements avaialble.

Young's modulus: MPa **Poisson ratio:**

The table below list all available data models and data sets pertraining to the delected material; the list is empty if no data or models are found in the database. The items in these lists are clickable!

Material Models			Experimental Data		
Microstructure	**Damage**	**Fatigue**	**Images**	**Stress-Strain**	**Strain-Life**
• 100x-Result	• Tom2 • TH_blind_T1R4N4_A1 • Mark	• MSF_result • SampleSavedResult	• image1.dat.dat	• AL7075_MSF.data • 1238447582547 • AL7075_MSF.data • 205Q-095_MSF.data • data1.dat • T1R4N4_A1_DMG.data1.dat • 99PCal_c1r4n3.a1 • T1R4N4_A1_DMG.data • fads	• AL7075_MSF.data • MyNewMSFDataset • 205Q-190_MSF.data

Figure A.5 Screenshot of summary page state

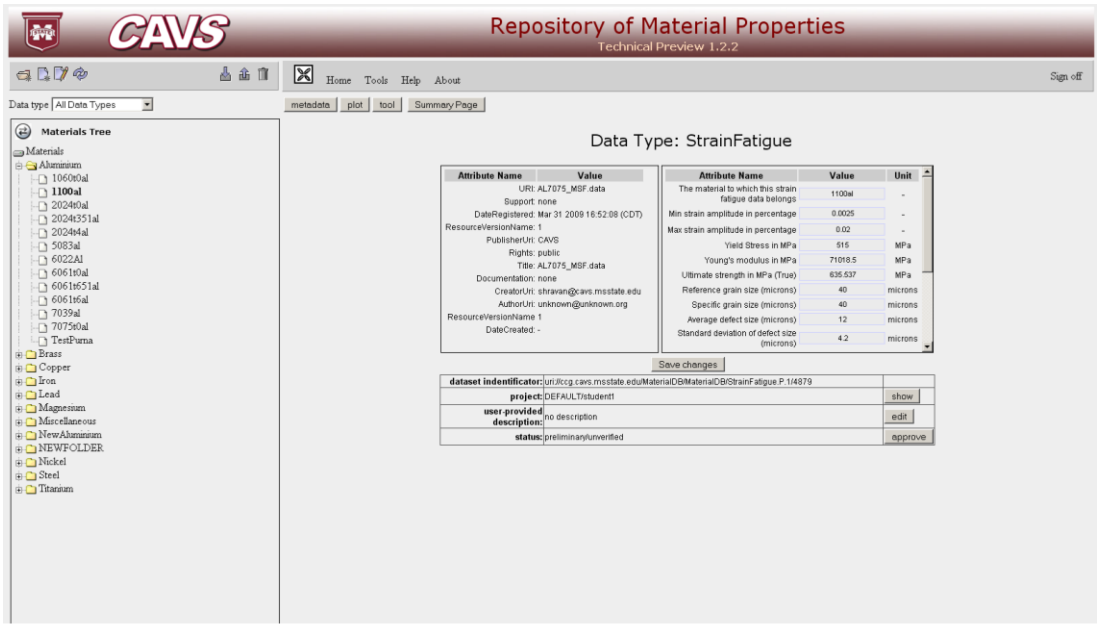
URI of summary page state:

uri://ccg.cavs.msstate.edu/Home/1489/all

Table A.5 List of containers and their values in summary page state

Container name	Banner	Button bar	Menu bar	Data Types	Action Button bar	Tree	Tree-2	Applet window
Value	Default	Default	Home	All data Types	Summary page	Material selected	Default	Show summary

Metadata state:



Data Type: StrainFatigue

Attribute Name	Value
URI:	AL7075_MSF.data
Support:	none
DateRegistered:	Mar 31 2009 16:52:08 (CDT)
ResourceVersionName:	1
PublisherUri:	CAVS
Rights:	public
Title:	AL7075_MSF.data
Documentation:	none
CreatorUri:	shravan@cavs.msstate.edu
AuthorUri:	unknown@unknown.org
ResourceVersionName	1
DateCreated:	-

Attribute Name	Value	Unit
The material to which this strain fatigue data belongs	1100al	-
Min strain amplitude in percentage	0.0025	-
Max strain amplitude in percentage	0.02	-
Yield Stress in MPa	515	MPa
Young's modulus in MPa	71018.5	MPa
Ultimate strength in MPa (True)	635.537	MPa
Reference grain size (microns)	40	microns
Specific grain size (microns)	40	microns
Average defect size (microns)	12	microns
Standard deviation of defect size (microns)	4.2	microns

dataset indentificator:	uri://ccg.cavs.msstate.edu/MaterialDB/MaterialDB/StrainFatigue.P.1/4879	
project:	DEFAULT/student1	show
user-provided description:	no description	edit
status:	preliminary/unverified	approve

Figure A.6 Screenshot of metadata state

URI of metadata state:

uri://ccg.cavs.msstate.edu/Home/4888/metadata/Fatigue.1

Table A.6 List of containers and their values in metadata state

Container name	Banner	Button bar	Menu bar	Data Types	Action Button bar	Tree	Tree-2	Applet window
Value	Default	Default	Home	All data types/ a particular data type from the list	metadata	material selected (material tree)/ dataset selected (project tree)	Dataset selected from the list/ default	Show metadata

Plot state:

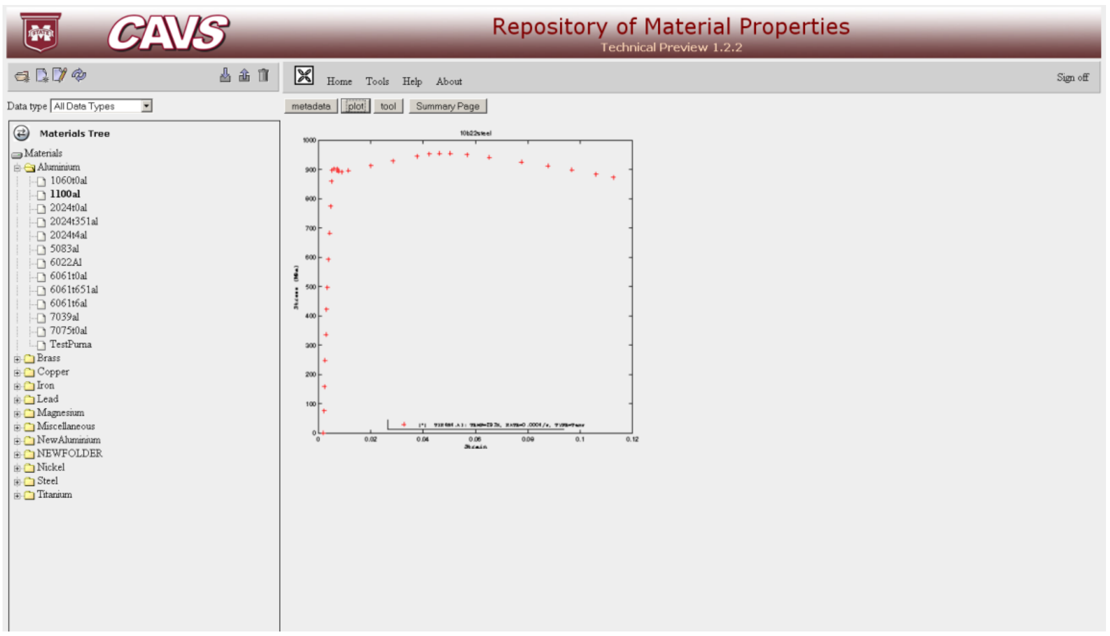


Figure A.7 Screenshot of plot state

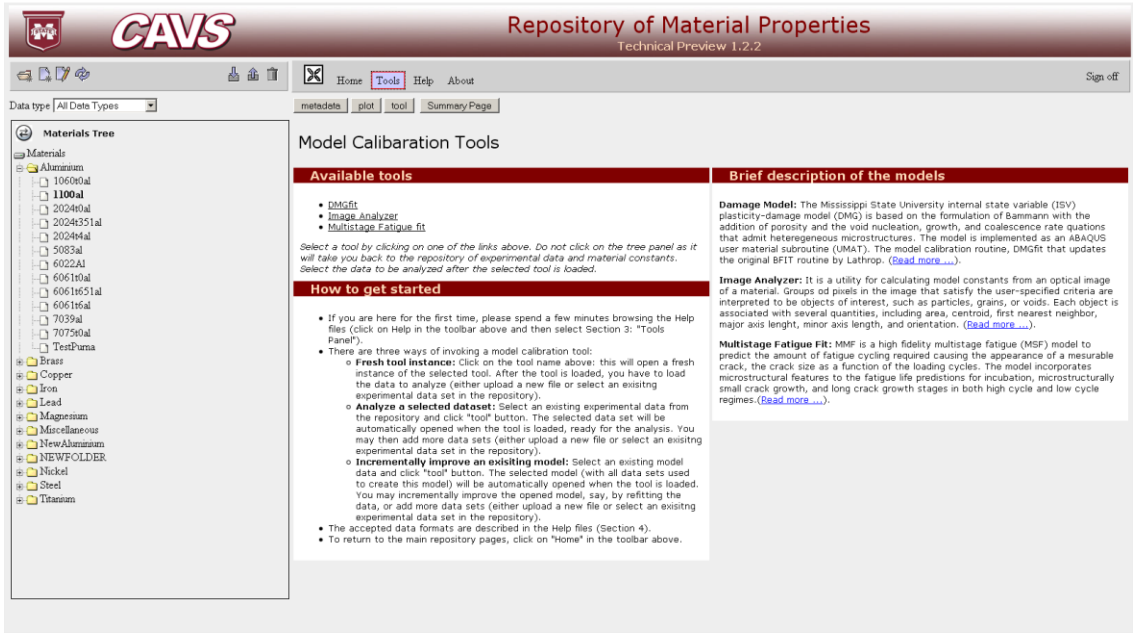
URI of plot state:

uri://ccg.cavs.msstate.edu/Home/4798/Plot/StressStrain.P.1

Table A.7 List of containers and their values in plot state

Container name	Banner	Button bar	Menu bar	Data Types	Action Button bar	Tree	Tree-2	Applet window
Value	Default	Default	Home	All data types/ a particular data type from the list	plot	material selected (material tree)/ dataset selected (project tree)	Dataset selected from the list/ default	Show plot

Tools state:



Model Calibaration Tools

Available tools

- DMGfit
- Image Analyzer
- Multistage Fatigue fit

Select a tool by clicking on one of the links above. Do not click on the tree panel as it will take you back to the repository of experimental data and material constants. Select the data to be analyzed after the selected tool is loaded.

How to get started

- If you are here for the first time, please spend a few minutes browsing the Help files (click on Help in the toolbar above and then select Section 3: "Tools Panel").
- There are three ways of invoking a model calibration tool:
  - **Fresh tool instance:** Click on the tool name above: this will open a fresh instance of the selected tool. After the tool is loaded, you have to load the data to analyze (either upload a new file or select an exisitng experimental data set in the repository).
  - **Analyze a selected dataset:** Select an existing experimental data from the repository and click "tool" button. The selected data set will be automatically opened when the tool is loaded, ready for the analysis. You may then add more data sets (either upload a new file or select an exisitng experimental data set in the repository).
  - **Incrementally improve an exisiting model:** Select an existing model data and click "tool" button. The selected model (with all data sets used to create this model) will be automatically opened when the tool is loaded. You may incrementally improve the opened model, say, by refitting the data, or add more data sets (either upload a new file or select an exisitng experimental data set in the repository).
- The accepted data formats are described in the Help files (Section 4).
- To return to the main repository pages, click on "Home" in the toolbar above.

Brief description of the models

Damage Model: The Mississippi State University internal state variable (ISV) plasticity-damage model (DMG) is based on the formulation of Bammann with the addition of porosity and the void nucleation, growth, and coalescence rate quations that admit heteregeneous microstructures. The model is implemented as an ABAQUS user material subroutine (UMAT). The model calibration routine, DMGfit that updates the original BFIT routine by Lathrop. (Read more ...).

Image Analyzer: It is a utility for calculating model constants from an optical image of a material. Groups od pixels in the image that satisfy the user-specified criteria are interpreted to be objects of interest, such as particles, grains, or voids. Each object is associated with several quantities, including area, centroid, first nearest neighbor, major axis lenght, minor axis length, and orientation. (Read more ...).

Multistage Fatigue Fit: MMF is a high fidelity multistage fatigue (MSF) model to predict the amount of fatigue cycling required causing the appearance of a mesurable crack, the crack size as a function of the loading cycles. The model incorporates microstructural features to the fatigue life predistions for incubation, microstructurally small crack growth, and long crack growth stages in both high cycle and low cycle regimes.(Read more ...).

Figure A.8 Screenshot of tools state

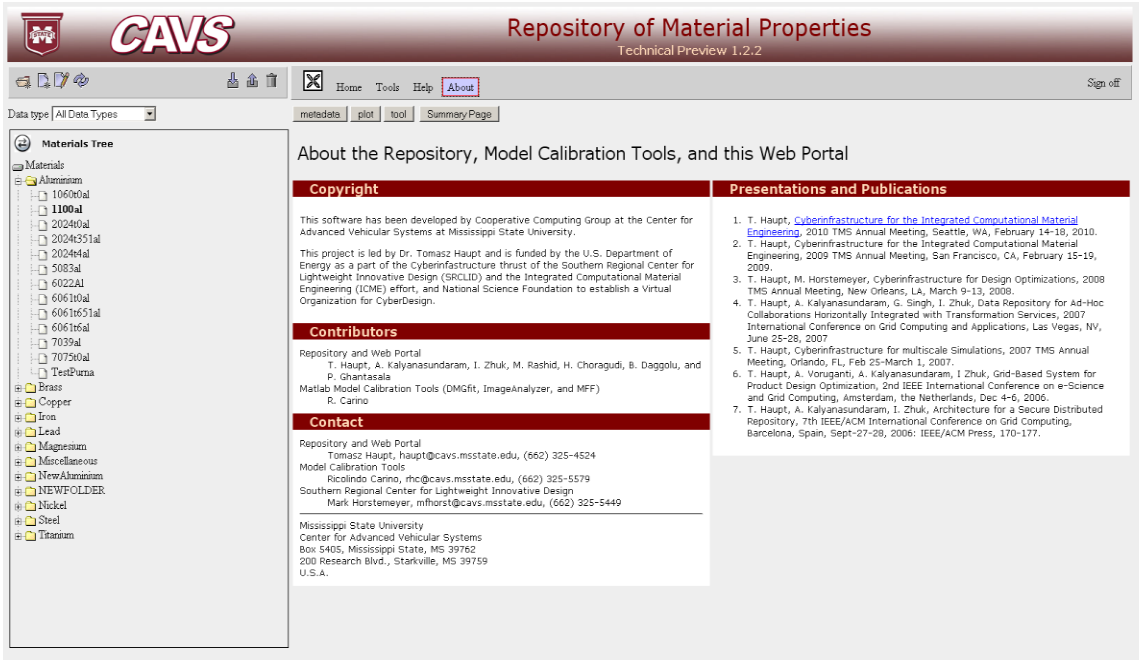
URI of tools state:

uri://ccg.cavs.msstate.edu/tools

Table A.8 List of containers and their values in tools state

Container name	Banner	Button bar	Menu bar	Data Types	Action Button bar	Tree	Tree-2	Applet window
Value	Default	Default	Tools	Any	Default	Any	Any	Show tools

About state:



About the Repository, Model Calibration Tools, and this Web Portal

Copyright

This software has been developed by Cooperative Computing Group at the Center for Advanced Vehicular Systems at Mississippi State University.

This project is led by Dr. Tomasz Haupt and is funded by the U.S. Department of Energy as a part of the Cyberinfastructure thrust of the Southern Regional Center for Lightweight Innovative Design (SRCLID) and the Integrated Computational Material Engineering (ICME) effort, and National Science Foundation to establish a Virtual Organization for CyberDesign.

Contributors

Repository and Web Portal
T. Haupt, A. Kalyanasundaram, I. Zhuk, M. Rashid, H. Choragudi, B. Daggolu, and P. Ghantasala
Matlab Model Calibration Tools (DMGfit, ImageAnalyzer, and MFF)
R. Carino

Contact

Repository and Web Portal
Tomasz Haupt, haupt@cavs.msstate.edu, (662) 325-4524
Model Calibration Tools
Ricolindo Carino, rhc@cavs.msstate.edu, (662) 325-5579
Southern Regional Center for Lightweight Innovative Design
Mark Horstemeyer, mfhorst@cavs.msstate.edu, (662) 325-5449

Mississippi State University
Center for Advanced Vehicular Systems
Box 5405, Mississippi State, MS 39762
200 Research Blvd., Starkville, MS 39759
U.S.A.

Presentations and Publications

1. T. Haupt, Cyberinfrastructure for the Integrated Computational Material Engineering, 2010 TMS Annual Meeting, Seattle, WA, February 14-18, 2010.
2. T. Haupt, Cyberinfrastructure for the Integrated Computational Material Engineering, 2009 TMS Annual Meeting, San Francisco, CA, February 15-19, 2009.
3. T. Haupt, M. Horstemeyer, Cyberinfrastructure for Design Optimizations, 2008 TMS Annual Meeting, New Orleans, LA, March 9-13, 2008.
4. T. Haupt, A. Kalyanasundaram, G. Singh, I. Zhuk, Data Repository for Ad-Hoc Collaborations Horizontally Integrated with Transformation Services, 2007 International Conference on Grid Computing and Applications, Las Vegas, NV, June 25-28, 2007
5. T. Haupt, Cyberinfrastructure for multiscale Simulations, 2007 TMS Annual Meeting, Orlando, FL, Feb 25-March 1, 2007.
6. T. Haupt, A. Voruganti, A. Kalyanasundaram, I Zhuk, Grid-Based System for Product Design Optimization, 2nd IEEE International Conference on e-Science and Grid Computing, Amsterdam, the Netherlands, Dec 4-6, 2006.
7. T. Haupt, A. Kalyanasundaram, I. Zhuk, Architecture for a Secure Distributed Repository, 7th IEEE/ACM International Conference on Grid Computing, Barcelona, Spain, Sept-27-28, 2006: IEEE/ACM Press, 170-177.

Figure A.9 Screenshot of about state

URI of About state:

uri://ccg.cavs.msstate.edu/About

Table A.9 List of containers and their values in about state

Container name	Banner	Button bar	Menu bar	Data Types	Action Button bar	Tree	Tree-2	Applet window
Value	Default	Default	About	Any	Default	Any	Any	Show about

A.7 Identify all the onlick events

The following table gives the list of all the onlick events that are present in the MPR that can cause a state transition:

1. Home tab selected
2. Tools tab selected
3. About tab selected
4. Material selected
5. Dataset in tree-2 selected
6. Dataset in summary page selected
7. Dataset in projects tree selected
8. Metadata button selected
9. Plot button selected
10. Summary page button selected

A.8 Save all the information related to the containers

The hash table can be used to save the information related to the containers when an onclick event occurs, and each entry in the hash table has information related to each state. Each time an entry is entered into the hash table the hash index is incremented by one. The hash table is shown in figure A.10.

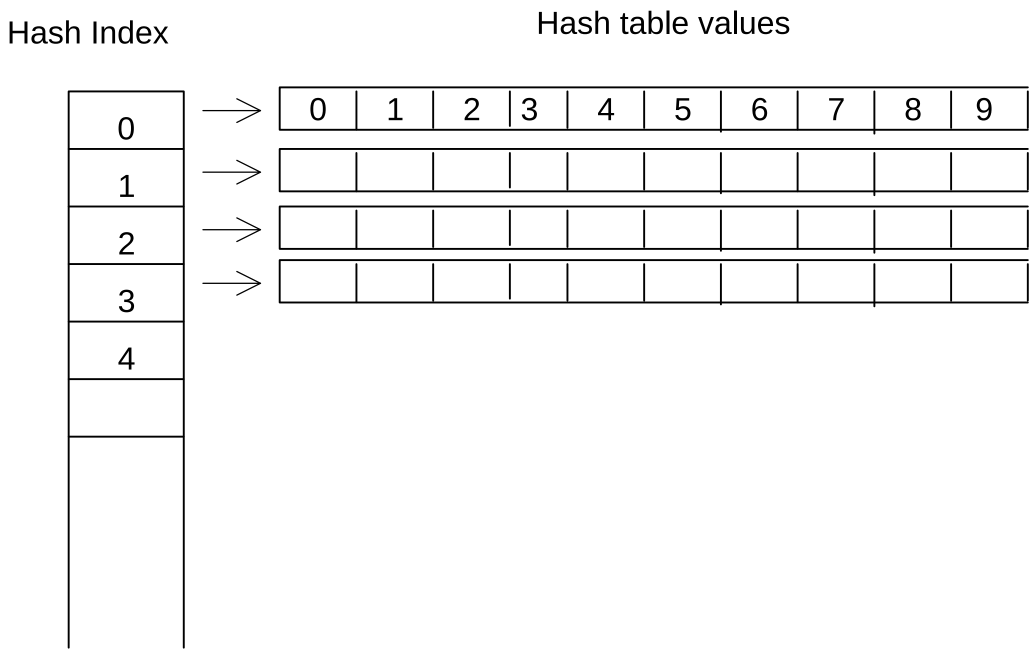


Figure A.10 Hash table to store values from SaveContainerValues()

Each entry in the hash table has 10 values in it, and each of these values refers to different aspects of the containers in the webpage. Table A.10 shows each entry and its corresponding information.

Table A.10 Hash table values and the corresponding variable

No.	Variable
Hashtable[Index,0]	Tab name
Hashtable[Index,1]	Data type
Hashtable[Index,2]	Action
Hashtable[Index,3]	Tree selected
Hashtable[Index,4]	Material Classes opened/Projects opened
Hashtable[Index,5]	MaterialURI/datasetURI
Hashtable[Index,6]	Tree-2
Hashtable[Index,7]	datasetURI selected in tree-2
Hashtable[Index,8]	Button
Hashtable[Index,9]	Dataset_Identifier

A.9 Create the URI by using the information from the hash table

Once the values related to the containers are saved, the URI of the state is can be created from the hash table by taking values from the entries in the hash table specified by the hash name, as shown in table A.11. This hash table returns the URI of the state.

Table A.11 CreateURI hash table for creating the URI of the state

Hash name	**Value**
Home	uri://ccg.cavs.msstate.edu/+Tabname
Tools	uri://ccg.cavs.msstate.edu/+Tabname
About	uri://ccg.cavs.msstate.edu/+Tabname
SummaryPage	uri://ccg.cavs.msstate.edu/Home/+Dataset_Identifier+/all
Metadata	uri://ccg.cavs.msstate.edu/Home/+ Dataset_Identifier +/metadata/+DataType
Plot	uri://ccg.cavs.msstate.edu/Home/+ Dataset_Identifier +/metadata/+DataType

A.10 Implement the design by using the URIs to get the states

A.10.1 Create URIs

The URI of the resource contains information related to each of the four containers that affects the content of the applet window. However, in some cases we can get the value of one container if we know the value of another container. For instance, if we know the URI of the dataset from the tree, then we can know the class version of the dataset from this URI without reading the datatypes container. Hence, we change the design to the one seen in figure A.11 from figure A.3, and we see that the values of the variables Dataset_Identifier and Datatype can be obtained from the function to determine dataset_Identifier and datatype values. Hence, the following URI format is selected. The algorithm for the function to determine dataset_Identifier and datatype values is shown in figure A.12.

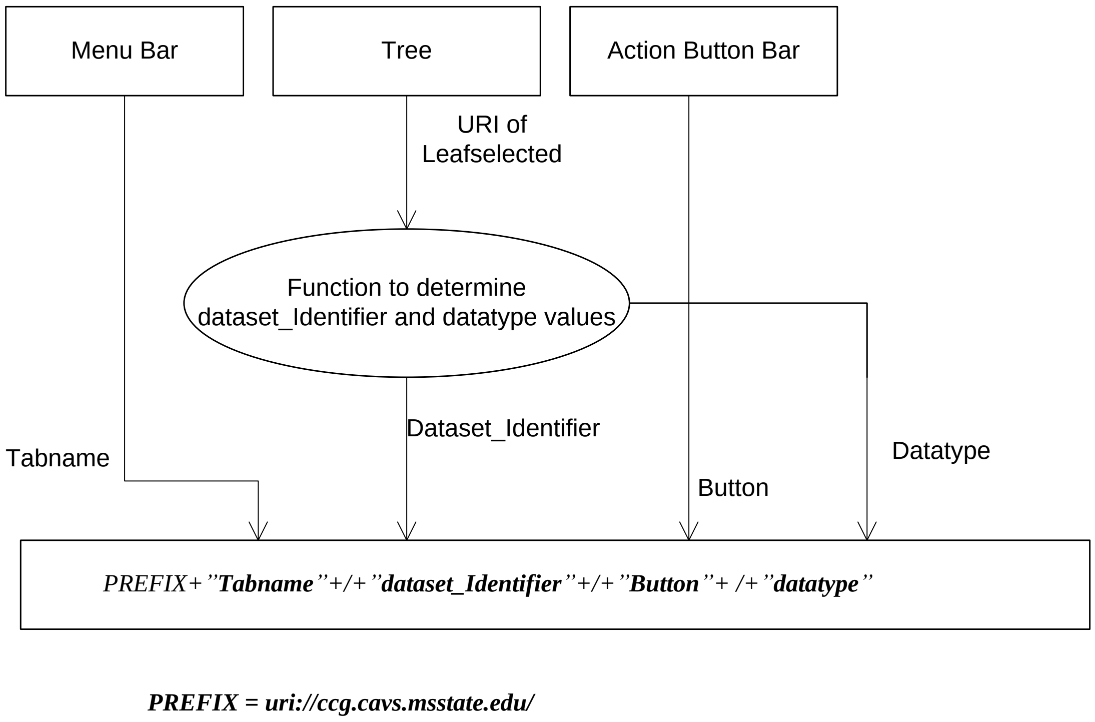

Menu Bar
Tree
Action Button Bar
URI of Leafselected
Function to determine dataset_Identifier and datatype values
Tabname
Dataset_Identifier
Button
Datatype
PREFIX+"Tabname"+/+"dataset_Identifier"+/+"Button"+ /+"datatype"
PREFIX = uri://ccg.cavs.msstate.edu/

Figure A.11 Creating a URI schema

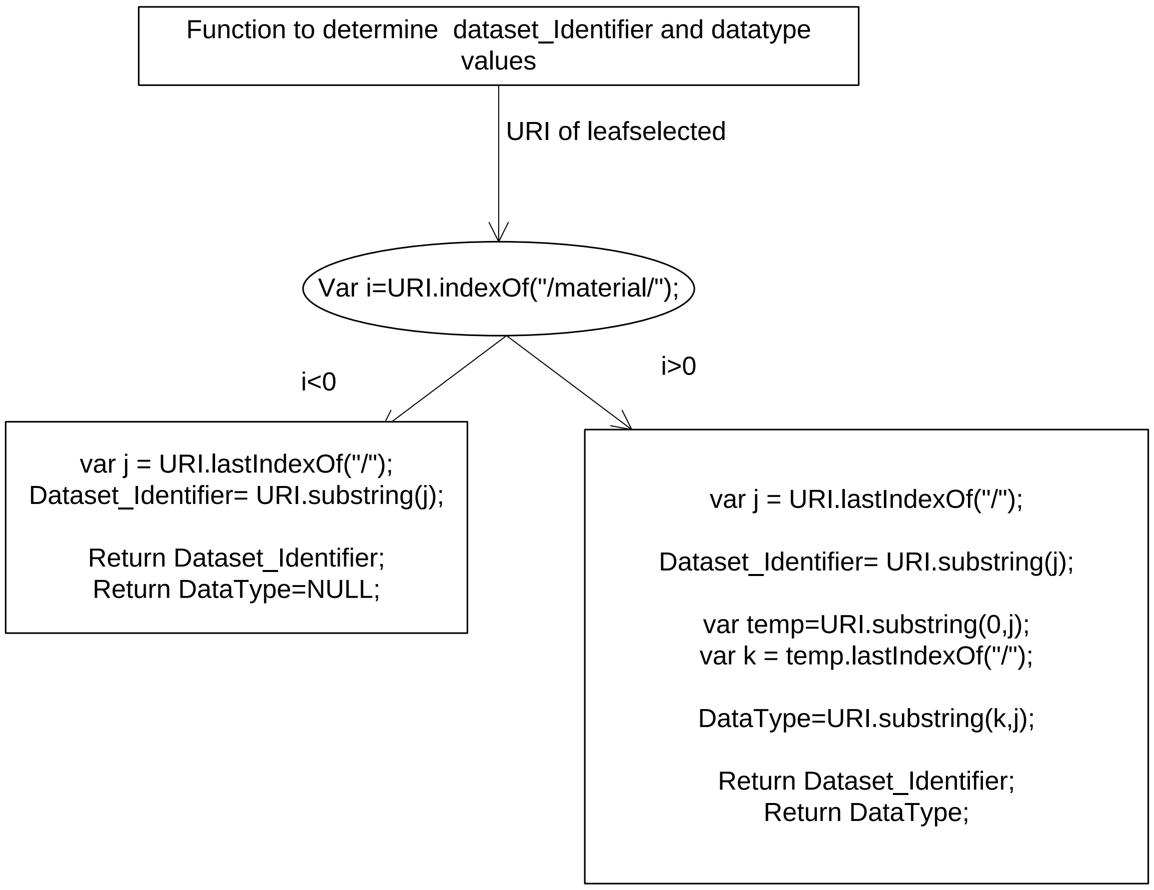

Function to determine dataset_Identifier and datatype values
URI of leafselected
Var i=URI.indexOf("/material/");
i<0
i>0
var j = URI.lastIndexOf("/");
Dataset_Identifier= URI.substring(j);
Return Dataset_Identifier;
Return DataType=NULL;
var j = URI.lastIndexOf("/");
Dataset_Identifier= URI.substring(j);
var temp=URI.substring(0,j);
var k = temp.lastIndexOf("/");
DataType=URI.substring(k,j);
Return Dataset_Identifier;
Return DataType;

Figure A.12 Algorithm for determining the Dataset_Identifier and DataType

A.10.2 Onclick event handling

The RESTful implementation of an AJAX-based webpage can be considered as a finite state machine moving from the initial state to the next state which is triggered by the state transition of one of the containers in the webpage. The events that cause the state transition are listed in the table A.12. When each of these events happen, we perform three actions, SaveContainerValues(), CreateURI() and ChangeState().

Table A.12 List of onclick events and their actions

Event	**Onclick action**

Home Tab selected	hashIndex=Call SaveContainerValues(),
	URI= CreateURI(Home,hashIndex)
	Call ChangeState(URI,flag=0, hashIndex)

Tools Tab selected	hashIndex=SaveContainerValues(),
	URI= CreateURI(Tools,hashIndex)
	Call ChangeState(URI,0,)

About tab selected	hashIndex=SaveContainerValues(),
	URI= CreateURI(About,hashIndex)
	Call ChangeState(URI,0)

Material selected	hashIndex= SaveContainerValues(),
	If(datatype=all){
	URI=
	CreateURI(SummaryPage,hashIndex);
	Call ChangeState(URI,0)}
	Else
	{load TREE-2}

Dataset in Tree-2 selected	hashIndex= SaveContainerValues(),
	URI= CreateURI(Metadata,hashIndex)
	Call ChangeState(URI,0)

Dataset in summary page selected	hashIndex=SaveContainerValues(),
	URI= CreateURI(Metadata,hashIndex)
	Call ChangeState(URI,0)

Dataset in projects tree selected	hashIndex=SaveContainerValues(),
	URI= CreateURI(Metadata,hashIndex)
	Call ChangeState(URI,0)

Metadata button selected	hashIndex=SaveContainerValues(),
	URI= CreateURI(Metadata,hashIndex)
	Call ChangeState(URI,0)

Plot button selected	hashIndex=SaveContainerValues(),
	URI= CreateURI(Plot,hashIndex)
	Call ChangeState(URI,0)

Summary Page button selected	hashIndex=SaveContainerValues(),
	URI=
	CreateURI(SummaryPage,hashIndex);
	Call ChangeState(URI,0)}

From the list of all the onlick events, we notice that the material selected event is inconsistent with the other implementations and stands out in the table A.12. The material selected event can have two possible actions based on the datatype selected. This shows that there is a need for better implementation of the summary page, which could be identified by designing the application RESTfully. The summary page is another implementation of the tree information; therefore, instead of having this information in the applet window, the tree container can be modified to accommodate this information.

The SaveContainerValues() function is responsible for collecting and saving the information related to all the containers of the webpage so that they can be used for bookmarking. The SaveContainerValues() uses the hash table to save the information; each entry in the hash table has information related to each state. Each time an entry is entered into the hash table, the hash index is incremented by one. The hash table is shown in the figure A.10.

A.10.3 Implementation of SaveContainerValues function

The SaveContainersValues() uses the hash table which is discussed in section A.8.

The following shows the pseudo code for implementing the SaveContainerValues.

```
SaveContainerValues()
{
Hashtable[Index,0]= Write code to get the tab selected

Hashtable[Index,1]= Write code to get the datatype selected from the drop down

Hashtable[Index,2]= Write code to get the action button selected

Hashtable[Index,3]= Write code to get the Tree cookie
Hashtable[Index,4]= Write code to get the Material Classes opened/Projects
opened
Hashtable[Index,5]= Write code to get the MaterialURI/datasetURI selected
Hashtable[Index,6]= Write code to get the Tree-2 cookie
Hashtable[Index,7]= Write code to get the datasetURI selected in tree-2
Hashtable[Index,8]= Write code to get the button selected from the button bar
Hashtable[Index,9]= Write code to determine the Dataset_Identifier.
}
```

A.10.4 Implementation of CreateURI function

Once the values related to the containers are saved, the URI of the state is generated by using CreateURI(Hashname,hashIndex). The URIs are created by taking values from the entry of the hashtable specified by the hash index, and the URI is formation as defined by the hash name as shown in table A.12. The CreateURI returns the URI of the state which should be loaded. So, the URI is passed to the ChangeState function which is responsible for changing the state of the applet.

A.10.5 Implementation of ChangeState function

The current implementation of the MPR is designed using Method Invocation (RMI), where each action performed on the MPR (like clicking a button) would invoke a method. This leads to too many method definitions that become too complex to maintain and redundant, but the RESTful design of the MPR has URIs of the resources when compared to the methods in the previous implementation. When a user clicks on a button in the MPR webpage rather than invoking a method, a URI is created, as described in table A.12, and passed on to the ChangeState function (figure A.13), which is responsible for getting the resource defined by the URI. Therefore, instead of calling different methods for different actions, in the new design for every action a URI is created and passed on to the ChangeState function.

The parameters that are passed to the ChangeState function are the URI and the Flag. The URI defines the state of the MPR, while the Flag defines whether the URI should be saved or used for loading it on the web client. When the Flag value is 1, then the bookmarking feature is implemented, and when the Flag value is 0, the applet window is loaded with the resource, which the URI defines.

The URI has the information related to the menu bar, datatype, action button bar, and the tree. Once the changeState function is called, the URI is split into tab name, dataset_identifier, button, and the datatype by using the URI.split("/") function. From the datatype value, the class version is established, and from the dataset_identifier, the URI of the material/dataset is recreated based on the datatype it belongs to. The content of the applet window is defined in terms of the combination of the values of tab name, button, URI of class version, and URI of the material/dataset. Depending upon these combinations, the applet is loaded with the correct resource by calling the appropriate function, as shown in figure A.13.

The function to establish classversion does so, based on the datatype, as shown in figure A.14. The function to create the URI of the material/dataset is responsible for adding the dataset_Identifier to the classversion, which gives us the URI of the material/dataset is shown in figure A.15.

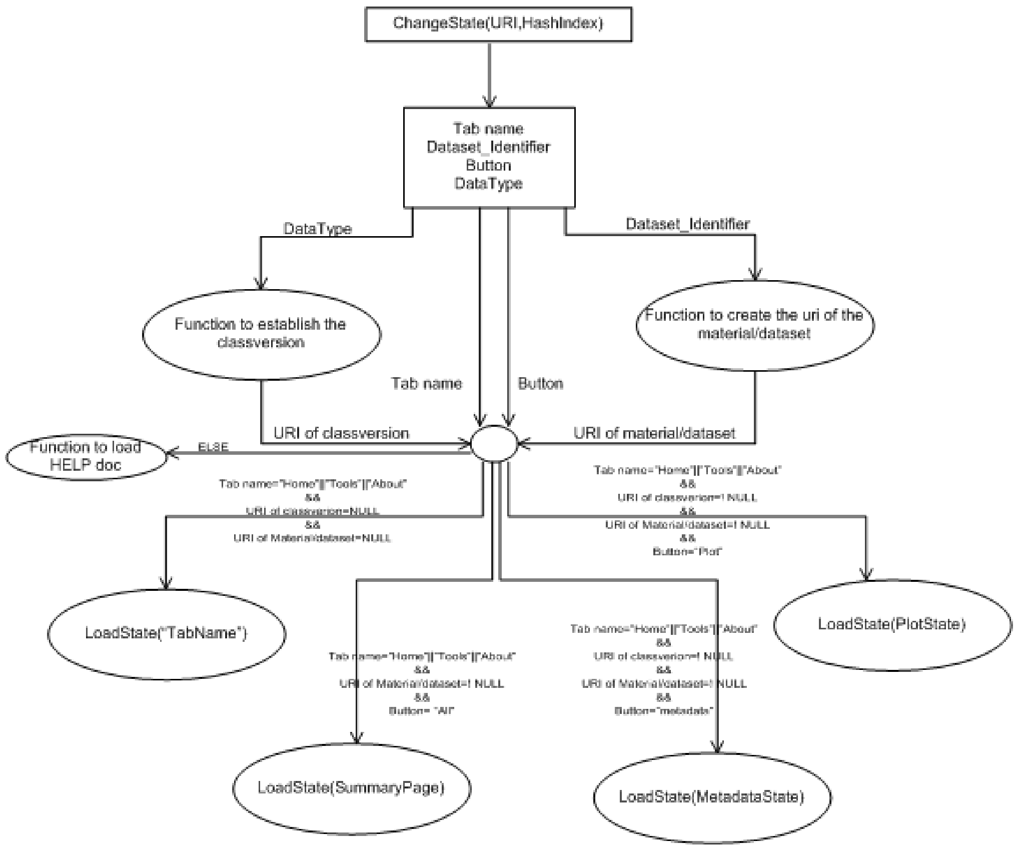

ChangeState(URI,HashIndex)
Tab name
Dataset_Identifier
Button
DataType
DataType
Dataset_Identifier
Function to establish the classversion
Function to create the uri of the material/dataset
Tab name
Button
URI of classversion
URI of material/dataset
Function to load HELP doc
ELSE
Tab name="Home"||"Tools"||"About" && URI of classverion=NULL && URI of Material/dataset=NULL
Tab name="Home"||"Tools"||"About" && URI of classverion=! NULL && URI of Material/dataset=! NULL && Button="Plot"
LoadState("TabName")
LoadState(PlotState)
Tab name="Home"||"Tools"||"About" && URI of Material/dataset=! NULL && Button= "All"
Tab name="Home"||"Tools"||"About" && URI of classverion=! NULL && URI of Material/dataset=! NULL && Button="metadata"
LoadState(SummaryPage)
LoadState(MetadataState)

Figure A.13 Control flow diagram of the ChangeState function

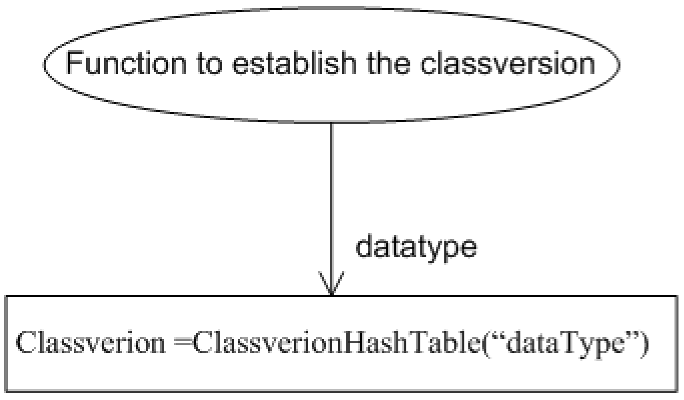

Function to establish the classversion
datatype
Classverion =ClassverionHashTable("dataType")

Figure A.14 Control flow diagram of the function to establish classversion

The function to establish classversion can be simplified by changing the way the datatype containers are handled in experimentalTypes.jsp. The current implementation of the datatypes container has a menu option value and menu option name. The menu option values are used to determine to the classversion, and the classversions are determined at several locations redundantly, which make the code very complex and difficult to maintain. This can be fixed by using the hash table called the ClassverionHashTable, which contains the menu option names and the classversions as the menu option values. This reduces the number of comparisons made in order to determine the classversion, and it also helps in extending to new classversions. If a new classversion is added to the MPR, then the change should be made only in the ClassverionHashTable (shown in table A.13), when compared to several locations in the current implementation. Problems like this can be easily identified and solved when designing RESTfully.

Current implementation of experimentalTypes.jsp

```
Data type <select id="dataTypeChooser">
<option value="AllDataTypes"> All Data Types </option>
<option value="StressStrain"> Stress-Strain </option>
<option value="Image"> Microstructure Image </option>
<option value="Fatigue_SL"> Strain-Life </option>
<option value="Damage"> Damage Model </option>
<option value="Microstructure"> Microstructure Model</option>
</select>
```

Redesign the experimentalTypes.jsp to the following:

```
Data type <select id="dataTypeChooser">
for(var i;i< ClassverionHashTable.length();i++)
){
<option value=" ClassverionHashTable .value[i]"> ClassverionHashTable
.name[i]</option>
}
</select>
```

Table A.13 ClassverionHashTable

Option name	**Option value**
All Data Types	All
Stress-Strain	uri://ccg.cavs.msstate.edu/MaterialDB/MaterialDB/StressStrain.P.1/
Microstructure Image	uri://ccg.cavs.msstate.edu/MaterialDB/MaterialImage/1/
Strain-Life	uri://ccg.cavs.msstate.edu/MaterialDB/MaterialFatigue/Fatigue.1/
Damage Model	uri://ccg.cavs.msstate.edu/MaterialDB/MaterialModels/Damage/
Microstructure Model	uri://ccg.cavs.msstate.edu/MaterialDB/MaterialModels/Microstructure/

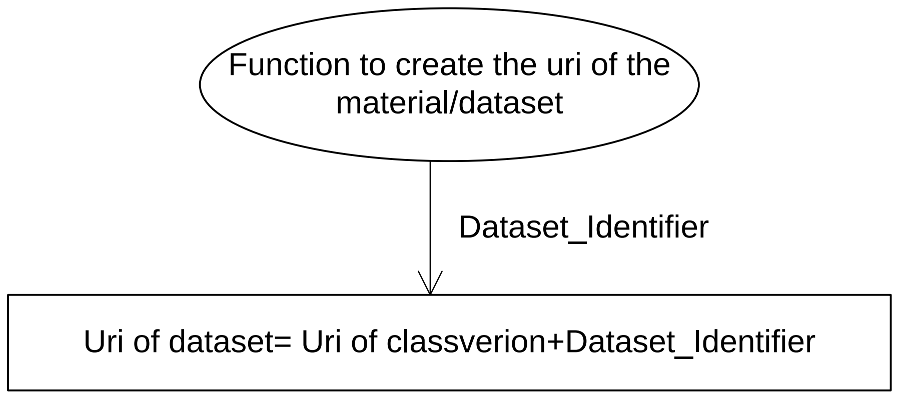

Function to create the uri of the material/dataset
Dataset_Identifier
Uri of dataset= Uri of classverion+Dataset_Identifier

Figure A.15 Control flow diagram of the function to establish Uri of dataset/material

A.10.6 LoadState function

From table A.14, we see that the LoadState hash table is responsible for loading the appropriate state based on the hash index values, which are home, tools, about, summary page, metadata, and plot. The LoadState hash table is shown in the table A.14.

Table A.14 loadState hashtable

Name	**Load applet**	**Function call (Javascripts)**	**Post processing (Function)**	**Additional calls**
HomeState	LoadApplet(Home)	setHome(Home)		setColor(Home)
ToolsState	LoadApplet(toolsHome)	setHome(tools)		setColor(tools)
AboutState	LoadApplet(About)	setHome(About)		setColor(about)
SummaryState	LoadApplet(materialSummary)	DisplayData(cv,id,dataType)	PostProcessing (Summary)	
MetadataState	MetadataHashtable(DataType)			
PlotState	PlotHashTable(DataType)			

The LoadState uses the LoadApplet hashtable for homeState, ToolsState, AboutState, and SummaryState. It uses the MetadataHashtable to load the MetadataState and PlothashTable to load the Plot state.

When the loadState index is homeState, ToolsState, AboutState, or SummaryState, the loadApplet (shown in table A.15) is used to directly load the state. When it is Metadatastate, the MetadataHashtable (shown in table A.16) is used to call the loadApplet hashtable, which makes decisions based on the datatype value. When the loadState index is PlotState, then the PlotHashTable (shown in table A.17) is used to call the loadApplet hashtable, which makes decisions based on the datatype value.

Table A.15 LoadApplet hashtable

Applet name	**Values**
Home	URL: home.jsp XSL: none Target: applet
Tools	URL tools/home.jsp XSL:none Target:applet
About	URL:about.jsp XSL: none target: applet
showImageModelMetadata	URL: getInstanceXmlByUri p_uri= + “uri of dataset” XSL: materalModels/modelMetadataImage.xsl target: dataActions
showDamageModelMetadata	URL: getInstanceXmlByUri p_uri=+ “uri of dataset” XSL: materialModels/modelMetadataDamage. xsl Target: dataActions
showMetadata	getInstanceXmlByUri p_uri XSL: experimentalData/showMetadata.xsl target: dataActions
showPlot	URL:PlotApplet.jsp XSL: Target: dataActions

Table A.16 MetadataHashtable

Name	**Load applet**	**Function call (Javascripts)**	**Post processing**
Microstructure	Loadapplet(showImageModelMetadata)	ProcessXMLInElementTags(xmlDoc)	PostProcessing(Microstructure)
Damage	Loadapplet(showDamageModelMetadata)	ProcessXMLInElementTagsDamage()	PostProcessing(Damage)
StressStrain	Loadapplet(StressStrain)		PostProcessing(StressStrain)
Image	Loadapplet(Image)		PostProcessing(Image)
Fatigue	Loadapplet(Fatigue)		PostProcessing(Fatigue)
MultiStageFatigue	Loadapplet(MultiStageFatigue)		PostProcessing(MultiStageFatigue)

Table A.17 PlotHashTable

Name	**Load applet**	**Function call (Javascripts)**	**Post processing (Function)**
StressStrain	Loadapplet(showPlot)	pasteAjaxResponse("/ccgportlets/apps/xfit/loadData.jsp",query,null,"dataActions")	
Image	Loadapplet(Image)	pasteAjaxResponse("/ccgportlets/apps/imageanalyzer/loadData.jsp",query,null,);	
Fatigue	Loadapplet(Fatigue)	getResponse("/ccgportlets/apps/msf/initModel.jsp") MSFModel.loadData(URI of dataset, 'true', function(resp))	

The loadState, MetadataHashtable, and PlotHashTable not only load the applets, but also have a function call and a post processing calls if needed, and the additional calls are the function calls which cause the side effect of the loaded applet. In the case of loading the tab states, setHome() function is responsible for clearing the global variables and then remembering the tab selected. The setColor() function is responsible for coloring the border of the tab which is under selected which is a side effect of loading the applet.

In the case of loading the summary states, DisplayData(cv,id,dataType) function is called, which is responsible for filling the columns in the summary page with the appropriate dataset names which are present in the material selected. The summary page requires post processing; hence, the PostProcessing() is called. The purpose of PostProcessing() is discussed later in this section.

In the case of loading the MetadataState, the decisions are made based on the datatype to determine which entry of the MetadataHashtable to load. If the datatype is Microstructure, then ProcessXMLInElementTags(xmlDoc) is called where xmlDoc=getResponseXML("/ccgportlets/rest/metadata?methodName=getInstanceXmlByUri&p_uri="+selectedDataItem), which is responsible for performing the XSLT transformation of the encrypted metadata into a tabular form. This state also needs post processing, so the function PostProcessing(dataType) is called.

If the datatype is Damage, then ProcessXMLInElementTagsDamage() is called, which is responsible for performing the XSLT transformation of the encrypted metadata into a tabular form. This state also needs post processing so the function PostProcessing(dataType) is called.

If the datatype is StressStrain, Image, Fatigue, or MultiStageFatigue then there is no functional calls made, because these states can be directly handled by the applet, but these states need post processing so the function PostProcessing(dataType) is called.

In the case of the plotState, the decisions are made based on the datatype to determine which entry of the PlotHashTable to be loaded. The plot state can be reached only through the metadata state, in order to go to the plot state. In the current design the loadApplet function is not called again, but it uses the applet that is already loaded. The plot image is pasted in the dataActions target of the “showMetadata” applet that is loaded in the metadata state. Therefore, no new applet is loaded in the function to handle plot. But, this is inconsistent to the design of each of the states and could cause problems while implementing bookmark and back button features. Therefore in the new design, a new applet is defined for the plot state called the “showPlot” which has the dataActions target in it where the plot image is pasted.

If the datatype is StressStrain, then the pasteAjaxResponse("/ccgportlets/ apps/xfit/loadData.jsp",query,null,"dataActions"); is called as shown in table A.17. The query is "?dataRef="+selectedDataItem, where the value of the selectedDataItem variable is equal to the URI of the dataset. The pasteAjaxResponse is responsible for pasting the plot image in the applet window placeholder (dataActions). There is no post processing for the plot state.

If the datatype is Image, then the pasteAjaxResponse("/ccgportlets/ apps/imageanalyzer/loadData.jsp",query,null,"dataActions"); is called as shown in table A.17. The query is "?dataRef="+selectedDataItem, where the value of the selectedDataItem variable is equal to the URI of the dataset. The pasteAjaxResponse is responsible for pasting the plot image in the applet window placeholder (dataActions). There is no post processing for the plot state.

If the datatype is Fatigue, then the getResponse("/ccgportlets/apps/msf /initModel.jsp") is called as shown in table A.17. The MSF tool was developed using DWR, so the mechanism of the handling the plot is different from Image and StressStrain. The DWR function MSFModel.loadData(dataRef, 'true', function(resp) is responsible for loading the plot image in the applet window place holder (dataActions);

here the value of the dataRef is the URI of the dataset. There is no post processing for the plot state.

Postprocessing function:

```
PostProcessing(name){
If(Summary){ set values of <young, poisson, materialName>}
If(Microstructure){set values of <model Type, Author URI, createURI,
DateRegisteredLabel, DateCreatedLabel, description>}
If(Damage){set the values of < model Type, Author URI, createURI,
DateRegisteredLabel, DateCreatedLabel, UriVal, description >}
If(StressStrain||Image||Fatigue||MultiStageFatigue){ set values of <data Type,
Author URI, createURI, DateRegisteredLabel, DateCreatedLabel, UriVal,
material, description >}
}
```

The PostProcessing function is responsible for updating some of the information in the loaded state of the applet.

A.11 Remember all the states of the FSM by remembering the URIs of the states

Once the MPR is designed RESTfully, the web-client can be considered as moving from one state to another as shown in figure A.2, where each state is identified by a unique URI. Therefore, in order to retrieve the previous state of the application, we should save the URI of the resource. When a user wants to get back to a previous state, then the saved URI is reloaded to reproduce the state. The client renders the information from the URI and then gets the resources that are represented by the URI to load the individual containers of the webpage. This concept of retrieving a state of an AJAX-based web-client can be demonstrated by designing the bookmark feature and back-forward button feature for the MPR, which is discussed in the following sections.

The current design of the MPR does not have the bookmark or the back-forward button features, but now that the MPR is redesigned RESTfully, the bookmark and back-forward button features can be implemented. So, the layout of the MPR should be added with a new menu bar, which looks like as shown in figure A.16. This menu bar has a bookmark button, a drop down menu to add the bookmarked URIs, a go button to load the bookmarked page, a back button, and a forward button.

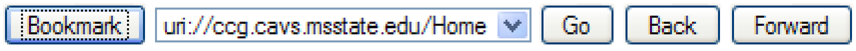


Figure A.16 layout of the bookmark and back_forward button bar

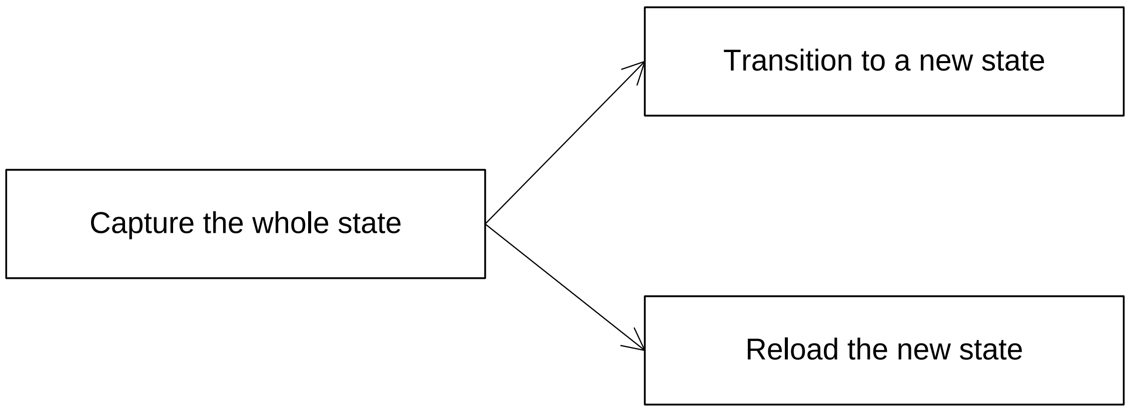


Figure A.17 Transition to new state

The figure A.17 shows different ways of achieving a new state. When the FSM is moving to new states by onclick events, then all containers need not be reloaded. Whereas when a new state is achieved by going to a bookmarked page or by a back button, then all the containers should be loaded to their previous states. The changeState() can perform the “transition to a new state”.

The “bookmark” button is responsible for saving the URI of the state of the MPR by calling the function Bookmarking(URI).

The “Go” button is responsible for reloading the state of the MPR defined by the URI, which is selected from the dropdown menu by calling the function ChangeState(URI,HashIndex) and RetrieveOtherContainersStates(HashIndex) is called to restore all the containers to the state indicated by the URI.

The “Back” button is responsible for loading the immediate previous state of the MPR by calling the function ChangeState(URI,HashIndex), where the URI is BackForwardButtonArray[INDEX--] and HashIndex is the index of the hash entry that contains all the information related to the state, provided we are not in the first state.

The “forward” button is responsible for loading the immediate next state of the MPR by calling the function ChangeState(URI,HashIndex), where the URI is BackForwardButtonArray[INDEX++] and HashIndex is the index of the hash entry that contains all the information related to the state, provided we are not in the final state.
This can be achieved by using the back-forwardButton(URI) function that keeps track of all the URIs in which the FSM is moving by saving the URI in an array, as shown in figure A.18

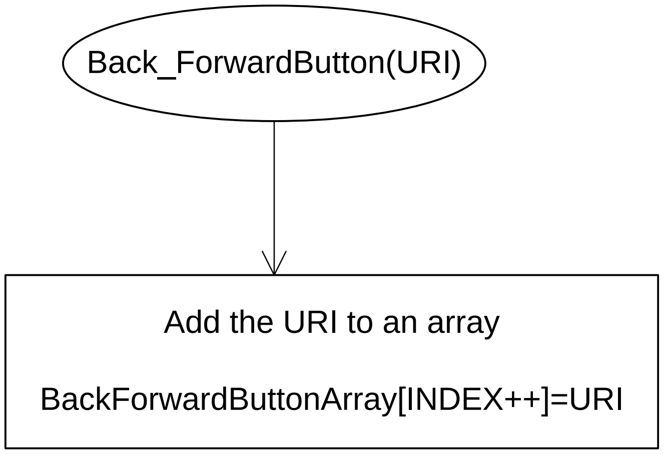


Figure A.18 Control flow diagram of the Back_ForwardButton function

A.12 Remember the URI of the state in order to bookmark a state

The Bookmark(URI) function that remembers the URI of the current state of the FSM by adding it to the bookmark drop down menu in an array is shown in figure A.19.

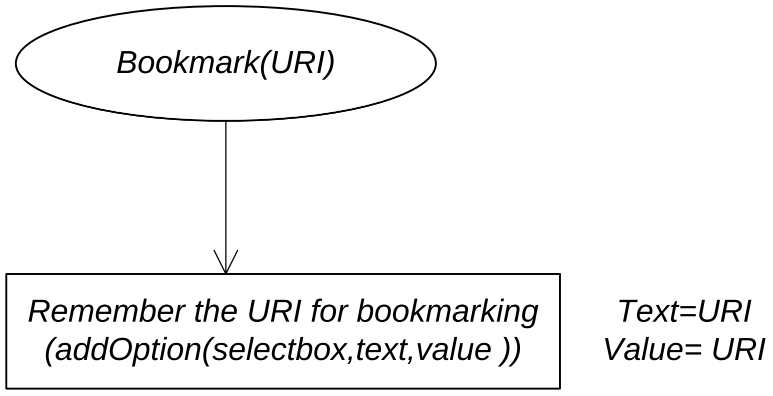

Bookmark(URI)
Remember the URI for bookmarking (addOption(selectbox,text,value))
Text=URI
Value= URI

Figure A.19 Control flow diagram of the bookmark function

In order to retrieve the previous state of the FSM, all the containers of the webpage should be set back to their respective previous states which are saved in the hashtable at the hashIndex. The RetrieveOtherContainersStates() function is responsible for doing this, as shown below.

RetrieveOtherContainersStates:

```
RetrieveOtherContainersStates(HashIndex){
If(Hashtable[Index,0]){Write code to highlight tab name which is in
Hashtable[Index,0]}
If(Hashtable[Index,1]){write code to display the datatype in the datatype
dropdown which is in Hashtable[Index,1]}
If(Hashtable[Index,2]){ write code to highlight action button which is in
Hashtable[Index,2]}
If(Hashtable[Index,3]){ write code to display the tree which is in
Hashtable[Index,3]}
If(Hashtable[Index,4]){ write code to open Material Classes opened/Projects
opened which are in hashtable[Index,4]}
If(Hashtable[Index,6]){write code to display the tree-2 which is in
Hashtable[Index,6]}
If(Hashtable[Index,8]){ write code to highlight the button selected which is in
Hashtable[Index,8]}
}
```

When the RetrieveOtherContainersStates(HashIndex) is called, the state of the webpage is retrieved by using the back button or by reloading a bookmarked state. So, when we try to retrieve the state of the entire webpage, the RetrieveOtherContainersStates(HashIndex) function is responsible for reading the hashtable entries and setting all the containers back to the previous state, as shown above pseudo code.