Application State Historian Architecture Draft

Curran Kelleher

Contents

1 Background Knowledge

1.1 List of Acronyms

- XML: eXtensible Markup Language
- RDF: Resource Description Framework (section 1.3)
- OWL: Web Ontology Language (section 1.4)
- OWL-DL: OWL Descriprion Logic (section 1.4)
- URI: Universal Resource Identifier
- URL: Universal Resource Locator
- IRI: International Resource Identifier
- HTTP: Hypertext Transfer Protocol
- MVC: Model View Controller
- GUI: Graphical User Interface
- ASH: Application State Historian (section 2)
- ASH-RE: ASH Runtime Environment (section 2.1)

1.2 The Semantic Web

The Semantic Web is a general vision for a world wide web whose distributed content is connected in a meaningful machine readable way. Originally, the Semantic Web was intended for use by annotating HTML documents to annotate their content. As the Web is becoming more data-centric, the usage of the Semantic Web for publishing and interlinking public database content. The primary Semantic Web technologies include Resource Description Framework (RDF) for encoding data and relationships, Web Ontology Language (OWL) for defining RDF vocabularies (RDF node and edge class hierarchies) and includes a description logic (DL) for expressing semantic constraints and axioms applicable to RDF graphs for validation and inference.

1.3 RDF: Resource Description Framework

RDF is a particular data model specification for of the semantic graph data structure, which is a directed graph with labeled edges. In RDF, identifiers for nodes and edges in an RDF (semantic) graph are URIs. RDF/XML is an XML format specification for expressing RDF graphs. In a particular architectural recommendataion for the Semantic Web by Tim Berners Lee called *Linked Data*, individual RDF nodes have URL identifiers which, when dereferenced (accessed via HTTP), yield descriptions of those nodes (incoming and outgoing edges) in RDF/XML format.

1.4 OWL: Web Ontology Language

OWL is a language used to describe RDF class and property hierarchies. OWL DL is a description logic (DL) for OWL.

1.5 MVC: Model View Controller

The Model View Controller (MVC) paradigm is a kind of software architecture in which the application definition is cleanly separated into three distinct components: the model, the view, and the controller. The model contains only a representation of the data structure which defines the application state. The view provides a GUI representation of the model. The controller is the bottleneck through which the view must go in order to modify the model.

1.6 The Singleton Design Pattern

The singleton design pattern is an object oriented software design pattern in which there is guaranteed to exists only a single instance (the singleton) of a class (the singleton class).

1.7 The Factory Pattern

The factory pattern is an object oriented software design pattern in which a singleton can be called upon to generate instances of a certain class. This pattern is typically used to enable dependency injection.

1.8 Dependency Resolution

Dependency resolution refers to the act of resolving the set of libraries that any given library ultimately depends upon. Maven and Ivy are examles of dependency resolution platforms.

1.9 Dependency Injection

Dependency injection refers to when managed software components depend upon runtime resources which are provided to them at runtime by the component manager. Typically the factory pattern is used to implement dependency injection.

2 ASH: Application State Historian

Application State Historian (ASH) is a software model for a plugin-based ontology oriented application architecture providing the following functionality: session history navigation (undo, redo, go to state), save application state, restore application state, synchronous collaboration (chat room style), and asynchronous collaboration (forum style). ASH is based on a combination of the the *semantic graph* data structure of RDF (section 1.3), the model view controller paradigm (MVC, section 1.5), and a plugin architecture which uses OWL (section 1.4) as a plugin capability description language. Dependency resolution (section 1.8) and injection (section 1.9) are also within the purview of ASH. ASH can be conceptually divided into three components: the runtime model, the plugin model, and the session history model.

2.1 Runtime Environment

The ASH Runtime Environment (ASH-RE) is a software model defining the achitectural structure of ASH runtime environment implementations, and can be characterized in terms of the model view controller paradigm (MVC, section 1.5). An ASH runtime environment supplies a general model and controller, and ASH plugins define views.

The *ASH runtime model* is an RDF graph which reflects the runtime state of the application, called the *ASH resource graph*. Each node in the ASH resource graph represents an *ASH resource*, which is a runtime software

component created by an ASH plugin and managed by the ASH runtime environment.

The ASH runtime controller provides an API for standard operations on the ASH-RM graph: create node (of a specified type), delete node, create edge (of a specified type), update edge and delete edge.

ASH-RM views are expected to be implemented by ASH plugins, which generate the runtime objects represented by nodes in the ASH-RM graph.

2.2 Plugin Model

The ASH Plugin Model (ASH-PM) defines the structure and function of ASH plugins. Each ASH plugin defines its own domain model using an OWL ontology, termed the *plugin domain ontology*. Every class in a plugin domain ontology must represent an ASH resource type, and every property must represent a property whose domain is a class in the ontology.

ASH plugins implement the factory pattern for ASH resources, but only for ASH resource types defined in their own domain models. The ASH runtime environment, when the create node operation is invoked, looks up the ASH plugin corresponding to the desired node type and calls upon it to create a new node of that type, which is comprised of two things: the node in the ASH-RM graph and a runtime object corresponding to that node. When operations are performed on that node, the ASH controller handles modification of the ASH-RM graph, and sends a notification of the change to the corresponging runtime object. This is how views get updated from model changes.

An ASH plugin domain ontology includes a URL from which its binaries can be downloaded. This enables *any* ASH session state to be restored at runtime in *any* ASH runtime environment connected to the Internet. This is however only possible when the plugin providers for all ASH plugins used by the particular session have published their plugins correctly, and that the particular ASH runtime environment is compatible with all plugins used.

It is expected that each ASH plugin provider make their plugins accessible in the following way:

1. Their plugin domain ontologies are published on the Internet.

2. The plugin provider has published their plugin domain model classes as Linked Data, meaning that the URI of each class is in fact a URL

which, when dereferenced (accessed via HTTP), yields an RDF/XML document describing that class, and thus containing the ontology URL.

3. The plugin provider maintains URLs from which their plugins can be downloaded.

4. The published plugin domain ontologies contain valid (live) plugin download URLs.

An ASH runtime environment can download and load all necessary plugins from the Internet at runtime. It does this in response to requests from the user to restore particular ASH session states - it must have downloaded and loaded all plugins which serve as factories for all resource types used in the session state. In this way, when a user attempts to load a state which contains ASH resources whose corresponding plugins are not currently loaded, the ASH runtime environment is able to dynamically download and load all binary code a state depends upon for the implementation of an ASH resource factory.

Instances of classes and properties from many ASH plugin domain models may exist simultaneously in a single ASH-RM graph. The range class of a plugin domain object property may be an ASH resource of a type defined in another plugin's domain ontology. This means that at runtime, the runtume object for a given ASH resource can manipulate other ASH resources from different plugins. In this case, the referencing plugin has a dependency on the referenced plugin. In this way, ASH plugins can capture the modular nature of software components.

2.3 Event Processing Model

The ASH event processing model is based on the notions of acomit actions, transactions and triggers. In ASH, an *atomic action* represents an invocation of an ASH controller method. In other words, each atomic action encapsulates an atomic change to the ASH-RM graph such as creating, modifying, and deleting ASH resources and their properties. A transaction represents a sequence of atomic actions whose consequences are only considered after the wntire sequence is applied to the ASH-RM graph.

ASH plugins can add arbitrary code to properties of particular ASH resources as triggers, which are executed at the end of a transaction containing

one or more atomic actions which modify those properties. This is advantageous over event listeners or direct callbacks because whereas callbacks and event listeners may be called numerous times in response to each atomic action, the trigger code is executed at most once per transaction.

2.4 Session History Model

The ASH history model is a session history graph in which nodes represent session states (snapshots of the ASH-RM graph) and edges represent state transitions (sequences of atomic actions which may span several transactions). Each state has a unique identifier. The present state of the ASH-RM graph corresponds to a state in the session graph, except when a new state transition is in progress.

In order to make the session history graph traversible, every state transition must have an executable inverse. Therefore the operations uncreate, undelete, and unset must be added to the controller API.

The ASH session component provides an augmented API which is a superset of the ASH-RM controller API. It adds functions relating to the notions of triggers, transactions, and state transitions.

2.5 Collaboration Model

The ASH collaboration model enables synchronous collaboration by framing the ASH-RM graph within a simplified distributed version control system. State transitions can be broadcast across the internet to other simultaneous users. The case of conflicting changes can be solved by rolling back the pending state transition in the client system which made the conflicting change (the one that arrived to the central collaboration server last).

The ASH collaboration model enables asynchronous by providing users access to the synchronous collaboration session graph, enabling them to "go back in time" and pursue alternate paths. Each concurrent synchronous collaboration session can be viewed as a session threadm having much the same branching structure as conventional forums.

2.6 ASH Applications

Since ASH is a generic software framework, it has tremendous flexibility in terms potential uses. This becomes especially apparent when one considers

that each resource generated by an ASH plugin may be a proxy for some object external to the ASH runtime environment, such as a graphics subsystem, a sound generation subsystem, input devices such as multitouch displays, and web resources such as communication channels (audio, video and text) and databases.

2.6.1 Food Inventory

One example application of ASH is a personal food inventory framework for handheld devices such as Android smartphones. A person using this application would be able to, using the touch screen interface, create, edit and delete: food class hierarchies (such as "bread" or "tea"), their own current inventory of food items (such as "a stick of butter in the fridge" or "three cans of soup in the pantry"), and recipes including ingredient requirements and cooking times. This would serve the purpose of an inventory tracking tool, but since the ASH model is an RDF graph, SPARQL or statements in OWL-DL could be applied to it in order to deduce answers to questions like "what can I have for dinner tonight that will take less than 20 minutes to prepare?" or "what meals can I make right now which will use up all my onions (which are about to go bad!)?".

2.6.2 Task Management

An ASH application for collaborative task management could be implemented within the ASH framework. The plugin domain ontology may include definitions of tasks, resources and processes along with the according logic which relates these entity types together. This application could provide a view with various facets such prioritized lists and Gantt charts.

2.6.3 Music Sequencer

It is concievable that an ASH plugin may provide the ability to manage local audio generation processes such as sequencers (and the patterns within), software synthesizers and samplers. In this scenario, a view could be created which allows the user to manipulate all sound generation components as well as the signal processing graph which connects them together in order to perform live computer music. When the synchronous collaboration facilities of ASH are applied to this scenario, it becomes possible to have a large group of musicians playing the machine at once. In addition, the players may be in

the same room, or they may be connected (via an ASH collaboration server) to players on the other side of the world.

2.6.4 Data Analysis

The data flow paradigm common to many data analysis packages (such as Weka) may be represented in the domain model of an ASH plugin. This data flow network may also be represented by an interactive view, allowing users to collaboratively analyze data.

2.6.5 Data Visualization

An collaborative interactive data visualization environment could be implemented in ASH. Each ASH resource could represent a visualization, including all parameters. Concievably, the input to visualizations could be the output from data analysis components defined in third party plugins.