THE INTRODUCTION OF HIERARCHICAL STRUCTURE AND APPLICATION SECURITY TO JAVA WEB START DEPLOYMENT

P. Karlsson, S. Jackson
CERN, Geneva, Switzerland

ABSTRACT

The Beam Diagnostics and Instrumentation software section is responsible for providing all the software necessary to develop, test, diagnose and maintain the different instruments produced for the CERN accelerator complex. Almost all graphical user interface applications are nowadays written in Java and the Java Web Start architecture is used to launch these applications. Java Web Start provides a solid basis for application deployment, but lacks several key features essential for ensuring a proper and easily manageable working environment (for example the lack of configuration templates and a security mechanism). This paper describes an internally developed tool, which extends the Java Web Start functionality in order to provide these missing features.

INTRODUCTION

The AB/BDI software section is responsible for providing all the software necessary to develop, test, diagnose and maintain the different instruments produced by the group. In fulfilling this responsibility, graphical tools ranging from test applications for hardware specialists to specialised applications used during machine developments are created for nearly all BDI instruments. Prior to 1997, most of these graphical based applications where written using the 'C' programming language and the MOTIF tool-kit under the Unix Xwindow system. Since the mid-nineties however, there has been a steady increase of users who prefer to run applications under the Microsoft Windows operating system, and there has been a consequent demand to have platform independent applications that will function on both the Unix and Windows platforms. The trend within CERN has been to develop Java based applications, and BDI is no exception.

Since the section started producing Java-based graphical user interfaces in 1997, the amount of Java-based software written within the BDI group has grown significantly. In fact, nowadays nearly all graphical user interfaces within the group are written in Java (a handful of specialist applications using LabView and other such proprietary development environments also exist). In the early days, the deployment of these Java applications was trivial (a single directory under which the compiled '.class' files where copied). As the complexity and number of applications grew however, access to the applications from a centralised source directory became untenable which led to the adoption of the industry standard Java web-start technology [1]. Java Web Start distributes applications from a central source (a web server), but each user's machine has a caching mechanism in which application files are stored. Each subsequent start-up of the same application uses these cached application files unless a newer version exists. This cuts down the need to download the application files over the network every time you start the application.

Although the use of Java Web Start aids in the deployment of applications, it doesn't help in the organisation of the applications. By the end of 2004, the AB/BDI Software section had in excess of 100 web-start files, accessible from a variety of different URLs. In addition, members of the section regularly use applications developed in other groups and rely on bookmarks to find the appropriate URL. Before the deluge of many more applications for the LHC era, it was decided to rationalise all these applications in a central place and have a group-wide, platform independent means to access all applications in a coherent way.

THE PROBLEMS

At the heart of Java Web Start lies the JNLP configuration file [1]. In addition to describing which resources should be downloaded in order to run a given application, it may also contain brief

descriptions and run-time parameters. Although the JNLP format describes individual applications very well, it doesn't address certain key problems found in BDI:

- *Repetition of common information* - The JNLP files for many BDI applications are almost identical apart from very slight modifications (different parameters for example) and maybe a few additional resources. To start each of these applications via Java Web Start involves duplicating common information in separate JNLP configuration files. A good example of this is the Expert GUI framework. This framework allows BDI software developers to develop expert applications using a pool of pre-made classes. The framework relies on many external resources (jar files for the middleware, graphing tools, etc) that need to be specified in the JNLP configuration file. Of course the framework isn't actually an application in itself, merely a mechanism for developers to launch their applications under. To use the framework however, developers must copy the mandatory framework resources and then add their resources for their specific application. This copying process is not only time-consuming but also leads to problems. When the framework designer decides to add or remove a resource from the framework for example, every single JNLP application file will have to be changed by hand.
- *Accessing CERN applications from other groups* - Java Web Start only allows a single location for browsing JNLP configuration files. In fact, BDI users access many other applications from other groups (such as timing diagnostics, middleware tools, FESA framework tools, etc).
- *Badly described applications* - The facilities to describe applications in the Java web-start configuration are very limited. For example, you can't see who is responsible for the application and which part of the accelerator chain it is used in.
- *Lack of security* - Security isn't addressed in the Java Web Start and anybody in CERN can run any application! In the case of sensitive applications working close to the beam, this is a serious issue.
- *Lack of application grouping* - The number of applications (internal and external) used within the group is set to grow rapidly as LHC approaches. Without any grouping or filtering of these applications, a user will find it difficult to find the desired application without traipsing through a long list. Furthermore, it may not be desirable that all users see a given application.

EXTENDING JAVA WEB START

It became clear that the Java Web Start tool and the JNLP format were not capable of declaring and representing BDI's applications. It was decided therefore to extend the Java Web Start tool and its JNLP syntax with an in-house application – The Application Launcher.

EXTENDING THE JNLP DECLARATION

The JNLP file format is well developed, universally accepted and widely used by industry. Although not totally compatible with BDI's needs, it was decided to extend this format rather than create a new one. Extension of the format implies wrapping the existing JNLP format inside our custom format. The JNLP syntax is XML, so wrapping an application's description simply involves embedding the <jnlp> tags inside our own proprietary XML file.

```
<jnlp>
  <resources>
    <jar href='http://webaddr/myfile.jar'/>
     …
  </resources>
  <application-desc main-class='toto'/>
</jnlp>
```

Figure 1: Original application description

```
<application-launcher>
  <application-group>
    <application>
      <jnlp>
        <resources>
          <jar href='http://webaddr/myfile.jar'/>
           …
        </resources>
        <application-desc main-class='toto'/>
      </jnlp>
    </application>
  </application-group>
</application-launcher>
```

Figure 2: Embedded application description

As you can see, the original JNLP description in figure 1 hasn't been modified, but has been embedded inside our extended tags. This means that we can now declare many applications in the same configuration file, and also introduce additional surrounding elements and attributes to further describe the applications, allowing filtering and grouping sets of applications later.

THE APPLICATION LAUNCHER SCHEMA

Although the basic structure of the JNLP launch file is declared by Sun in a DTD [1] (appendix C), launch files are never validated against it. If there are any problems, they are reported by the Java Web Start launcher during application launching. The Application Launcher configuration file however, is strictly validated against a schema, which contains the original DTD description (ported to the Schema language) as well as BDI specific tags used for filtering and grouping. Validating the configuration file in this way ensures that when an application description is modified or a new application is added to the configuration file, developers can't forget to declare some mandatory element or attribute for example. Furthermore, as the configuration file is well defined and well structured, we can use readily available generic tools that understand the Schema language to provide generic editing facilities and validation for free.

THE APPLICATION LAUNCHER GUI

The standard GUI shipped with Java Web Start is clearly incapable of understanding our extended configuration file. Luckily however, the Java Web Start application can be started both interactively (displaying a GUI), or via the command line. When started from the command line, the Java Web Start application accepts an argument indicating the location of the desired application's JNLP file. The Application Launcher exploits this feature by allowing the user to select a desired application from the centralised configuration file, extracting the contents of the selected application tag (i.e. the JNLP tag) and saving the contents to a temporary file. Java Web Start is then invoked and supplied with the temporary file as an argument thus allowing the Application Launcher to *fake* real JNLP files. The Application Launcher can be started from a link on the AB/BDI/SW section web page, or by running a Unix script.

In addition to simply starting individual applications, another requirement was to allow advanced filtering of applications and provide some form of security. The Application Launcher does this by providing several attributes used to filter the applications displayed and also limit the access to certain users. See figure 1 for some examples of extra attributes used.

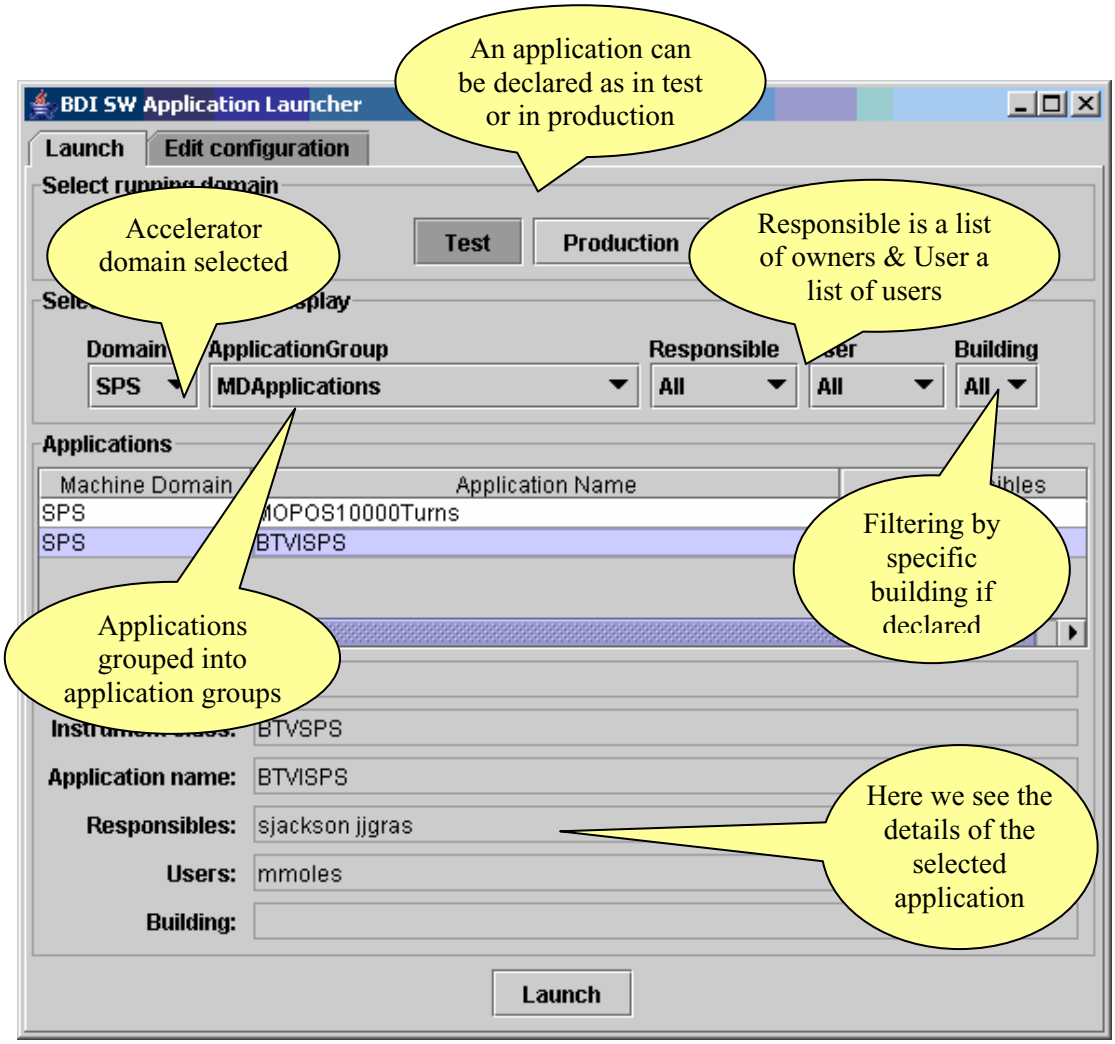

Figure 1: Snapshot of the Application Launcher showing the filterable attributes

Most of the attributes defined in the application launcher's configuration file are filterable in the user interface. The current user is also determined by the user interface and is used to provide security based on the 'Responsible' and 'User' attributes of a given entry in the configuration file. In fact, if the current user of the application isn't found in the *Responsible* or *User* list, the application will not even be displayed.

THE TOTAL ERADICATION OF JNLP FILES

Given that the Application Launcher provides a user-friendly way to configure and start Java web-start applications, the natural progression is to attempt to eradicate external JNLP files altogether. Of course references to external JNLP entities (from other groups) is beyond our control, but for BDI applications it should be possible to define all our applications via the tool. There is however a drawback to this. Most users prefer to start the applications via a link in a web page (traditionally to the JNLP file). Insisting that users start up an additional tool and search for the application from a potentially large list is both annoying and considered to be a 'step back'. In order to compete with and thus eradicate the need for JNLP files, the application launcher needs to be able to be started with a predefined application to launch.

Passing a parameter to the application launcher is not as trivial as it may first seem. The application launcher is itself started via a JNLP file which means that in order to pass an argument to the application, the contents of this JNLP file would have to be altered to include a property (containing the application to start). However, this means that developers would have to create a unique JNLP file in order to directly start their applications (copying the contents of the normal Application Launcher JNLP file and setting the property to their application) which defeats the object of the tool! To overcome this, it was decided to *serve* this application launcher JNLP file from a PHP script. This script reads the application launcher's standard JNLP file, inserts the given *launch* argument as a parameter and then returns the JNLP content to the calling browser. Using this script, a developer can directly invoke their application from a HTML link like this:

```
http://bdidev1/bdisoft/development/applauncher.php?launch=ActionBuilder
```

In this case, the applauncher script is invoked with the 'launch' variable set to 'ActionBuilder'. The script then loads the standard JNLP script, inserts the appropriate property to 'ActionBuilder', and then returns the JNLP content. The result is a dynamically mutating JNLP script driven by a simple parameter given in the URL. The 'ActionBuilder' parameter used in this example is the name of the application that you want to start.

Eradicating external JNLP files and forcing everything to be started through the tool is not only convenient but also introduces features supplied by the tool. Such an important feature given by the tool is security. In the past, anybody within CERN could start any BDI application given its link to the JNLP file. Eradicating the JNLP file and forcing the user to pass through the tool means that the access control mechanism is still in place. In other words, even though everyone can see a link to start an application, only those declared in the tool as 'users' can actually start the application. In the coming years many sensitive applications will be developed, potentially controlling instruments close to the beam. This indirect method of application invocation allows some level of security to protect these applications (and thus the equipment behind) from inquisitive intruders.

CONCLUSION

The Application Launcher has been designed specifically to fulfil our needs for Java application launching. As such, it supplies exactly what is required to classify, describe and start the applications used to control and diagnose BDI's instruments. Given that it is based on the standard Java Web Start technology, it should be future-proof and needs only to evolve with the JNLP format, which seems very stable. Given that the Application Launcher is a standalone Java application, it allows users to start the applications from both Linux and Windows seamlessly and doesn't necessarily involve the configuration of browsers (as would be the case with a HTML-based navigation tool). The configuration of the Application Launcher is done in a standardised way and makes use of the commonplace XML language, strictly governed by the Schema declaration. This means that although editing facilities are built into the application, we don't need to rely on any in-house editor. The resulting application will allow the BDI group to rationalise and classify accurately, each of its applications while at the same time providing a level of security not previously available.

REFERENCES

[1] Java Web Start specification - *http://java.sun.com/products/javawebstart/download-spec.html*