Project Report

Mission Planning and Sequencing Investigation of Third Party Software

Mike Mozingo (Space Grant) Jet Propulsion Laboratory, California Institute of Technology

Barbara Streiffert (Mentor) Jet Propulsion Laboratory, California Institute of Technology

Dennis Page (Co-Mentor) Jet Propulsion Laboratory, California Institute of Technology

August 11, 2011

Introduction

Mission Planning and Sequencing (MPS) maintains a system called the Automated Sequence Processor (ASP) which is responsible for checking non-interactive commands and preparing them for radiation to spacecraft. In order to streamline the process and increase maintainability MPS is looking to use a third party workflow engine to control the ASP. In addition to increasing productivity, another driver for the workflow paradigm is the new way that the software is going to represent the spacecraft state. The spacecraft state is going to be represented by a timeline data structure.

Background

The Mission Planning and Sequencing (MPS) element of the Multi-Missions Ground System and Services (MGSS) provides space missions with software that meets the demands of multiple missions. The requirements on this software set are to be able to plan spacecraft activities, sequence commands to the spacecraft, package the commands and execute these products on the spacecraft. MPS is looking to improve their multi-mission software to meet the demands of current and future missions in ways that are more cost effective and at the same time reduce the risk to the spacecraft.

One of the ways MPS is working on this task is with SEQ revitalization. SEQ revitalization includes many improvements to the current process; however the work this summer was for the ASP and timeline portion of the revitalization. The timeline data structure is a new paradigm for how to plan and store spacecraft operations. The current ASP uses files to represent commands which a spacecraft operations team member then pushes through multiple scripts to check the commands to ensure that they are Non Interactive Commands (NIC), constraint checks them, and then translate them into the binary representation that the spacecraft is expecting. The timeline would transform this activity into a central database where all data related to spacecraft operations are stored. This new paradigm would provide an intuitive way to represent operations with items such as a power graph that could show the power load on the spacecraft among other readouts. The operator then would generate a file to upload into the timeline and then would run different services that the timeline provides to convert the command for the spacecraft. This new way for dealing with commands will require a new way of thinking about the ASP.

MPS hopes to increase productivity by using a workflow engine to automate ASP processes. A workflow is a business process that describes a repeatable task that can be of varying complexity. Workflow engines are software that can design, create and orchestrate workflows. The engines often use a standard way to represent the workflows such as the Business Process Execution Language (BPEL).

Objectives

The ASP is currently a collection of a hundred or more scripts, written in various languages. These scripts provide all of the function that is required to take a command from genesis, to be ready to transmit to the spacecraft. This method is difficult to maintain, and does not make sense with the new timeline service paradigm. The task this summer was to investigate the possibility of using a third party workflow engine to replace the ASP, and control calls to the timeline service. To accomplish this task, there was a procedure to follow. First, requirements on the workflow engine must be created. This criteria would ensure that the engine would be suitable for JPL and as well as replicating the current functionality of the ASP. After the criteria were determined, then the engines in question would be graded against the requirements, and the best engine would be selected. Finally, a prototype workflow would be designed, implemented and tested.

1. Identify criteria used to determine trades on workflow engines
2. Select a workflow engine based on criteria
3. Create a prototype workflow of the ASP process using software stubs
4. Make the prototype extensible in order to communicate with the SEQ Revitalization Timeline database

Approach

The approach to this task follows the objectives list. The first step has been to define the requirements on the workflow engine. Some requirements were already posed such as the engine must be able to have process authoring by graphical tools. Process authoring is how the workflow is created either in the BPEL language (much like XML), or using graphical tools. The graphical tooling is one of the most important requirements in creating the workflow and allows the user to define the process using a series of blocks and arrows, which is a very intuitive way to represent the process. Another requirement was that the engine must have a plugin for Eclipse. A native plugin is preferable as it would integrate the best with all of the features of the engine, however because the processes are defined using standard languages there are many plugins that will work to modify the process flow.

In addition to the tooling requirements, there are two interface requirements. The first is that the workflow engine should have a web interface or console. This requirement would allow for a user to use their browser to initiate the web service and be able to control processes. While JPL could create a custom web service to handle this requirement, having one that comes with the engine would free developers to work on the ASP workflow instead of spending time on the front end web service. Furthermore, the engine should have a Representational State Transfer (ReST) interface to the processes. Having a ReST interface would allow for custom software to control the flow of the process. A

ReST interface feature would also let one process to control another process through the ReST Application Programming Interface (API). One example is that it could be used for queuing.

The final group of requirements on the engine was the set of requirements for the community. The current ASP has been used for about fifteen years and in general, software at JPL must be able to last a long time. This need leads to an open source options. Open source projects are supported by the community that first created them, and so as long as there is a community, the software remains viable. . Having an open source community can typically last longer than many businesses last or longer than a business will support a particular piece of software. However, open source software is only as strong as the community backing it which leads into the next requirement; there must be strong community support and documentation for the workflow engine.

Once the requirements on the engine were defined, several engines were judged against them.

	jBPM	Activity	Apache ODE	Intalio	Open Business Engine	Open For Business	JFlower	Pegasus
Graphical Tooling	Y	Y	Y	Y	N	N	?	Y
Eclipse Plugin	Y	Y	Y	N	N	N	?	?
Customization	Y	Y	Y	N	Y	Y	?	?
Web Interface	Y	Y	Y	Y	N	N	?	?
ReST API	Y	Y	Y	Y	N	N	?	?
Open Source	Y	Y	Y	N	Y	Y	?	Y
Documentation	Y	Y	Y	Y	Y-	Y-	?	Y
Community	Y	Y	Y	Y	N	N	?	Y

Table 1: Comparison table for workflow engines investigated.

Notes:

jBPM came with everything that was needed to get started.

Activity had dependency conflicts with Eclipse Indigo.

Apache ODE did not come with its own server but runs on a tomcat server.

JFlower only had a sourceforge download page.

Pegasus only ran on unix based systems.

From table 1 there are three engines that meet the specified criteria: jBPM, Activity and Apache ODE. From those, jBPM was chosen, however all would make excellent choices. Activity was not chosen because currently there is a dependency conflict while trying to install it in Eclipse Indigo. This dependency will likely be fixed in the future, but as of the time of this report it would not install. Apache

ODE, on the other hand, did not come with a server packaged with it. It is designed to be run on a Tomcat server. This left jBPM whose installer was very complete and came with everything that was necessary to create and deploy a process.

jBPM has several merits that make it a good choice for a workflow engine. First, it is well supported. The engine is made by Red Hat, and is mature with a major version of jBPM of 5. There is documentation for users and developers, along with a forum for users. The installer package comes with everything that is required to begin developing, including a JBoss server and a plugin for Eclipse [Fig. 1]. jBPM out of the box runs on a JBoss server and can also be configured to run on a Tomcat server. It may also be possible to run jBPM on a Glassfish server but that is less documented. Developing a process in jBPM is very customizable. It is possible to create a custom task and deploy it on to the server. Such a task is implemented in Java, so everything that Java can do is a possibility.

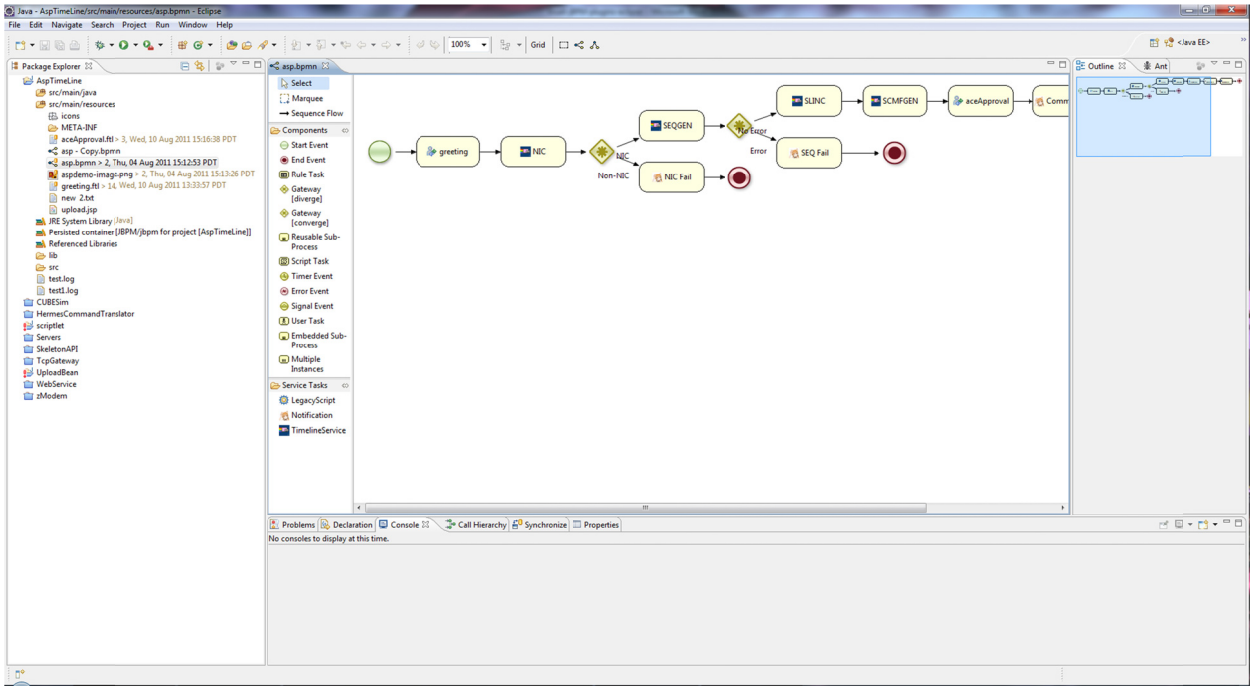

Figure 1: The jBPM process authoring plugin for eclipse.

Once the third party workflow engine was chosen, a sample process was created to demonstrate key functionality and as a proof of concept to show that the engine met all requirements for the new ASP paradigm.

To make a sample workflow first a few example projects were created to experiment with the process, eclipse plugin and the server interface. Next a custom task was created that could handle ReST interface calls. This task was tested inside of eclipse to verify functionality. To test the task, a webserver hosted on Glassfish was running on localhost. After the task was verified, a process that represented the workflow of the current ASP was created [Fig. 2]. This process ran under eclipse, and was then deployed to the server. The workflow was tested and found to be successful on the server.

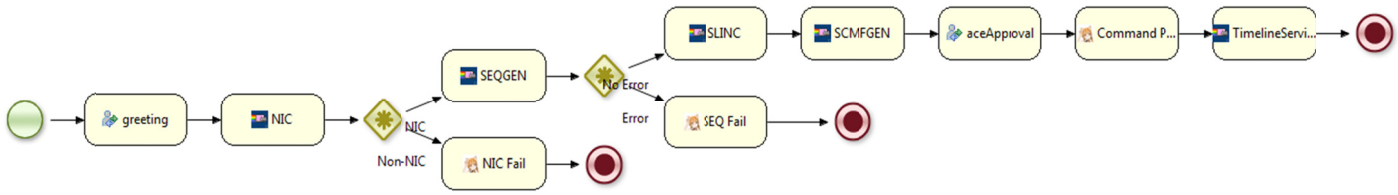

Figure 2: Workflow for the ASP process.

Results

The findings of this project indicate that the use of a workflow engine could increase productivity and maintainability for the ASP. The limitations of the current ASP hinder progress. Using a workflow engine, the process can be refined to a simple, yet fully capable state. This new workflow would integrate easily into the timeline paradigm, and would decrease errors due to bugs in the code.

Some concerns were raised about the ability of operators to modify the code quickly during runtime to fix problems encountered when trying to process commands. This ability is available in the current ASP. Since the current ASP is a collection of scripts, it is easy to look at an individual script and change some functionality quickly. This capability might be necessary during the early stages of the workflow testing but the idea of the design is that is can be constructed carefully to minimize bugs and keep the process running. There are also ways to mitigate errors during runtime with error handling branches and sub-processes in the flow. Through these methods it is possible to have a workflow that is easy to maintain and can handle all of the processing for current and future missions.

Discussion

All the tools functioned as advertised while testing jBPM. The user guide does a complete walkthrough of creating a simple process, and has examples of creating custom (domain specific) tasks. The guide also includes instructions for using Guvnor, JBoss' revision control software. Guvnor can be used to deploy processes to the server, as well as track revisions.

It was found that some additional jars created conflicts when using them in processes that were deployed to the server. The problem was likely a conflict with existing classes that were duplicated in the external jars (the Jersey API jars). This conflict resulted in the web console failing to load. To debug such a problem, JBoss records log data for the servers.

Another issue was that it was unclear at first was how to deploy custom work item handlers to the server. These handlers were part of the domain specific tasks, and were necessary for the sample process created. The jBPM forums are an excellent source of information and there are many posts giving guides for working with jBPM and JBoss.

One thing that would have really helped in developing the process would be to read both the jBPM user guide and the JBoss user guide. The jBPM guide does not cover all of the necessary information to get

started using the JBoss server and console. Most of the examples in the jBPM guide are geared towards working inside of Eclipse.

Conclusion

After an analysis of workflow engines and creating a sample process in jBPM it has been shown that the use of workflow engines could benefit the current ASP process and could be extended to support the Timeline Service and future versions of the ASP. While jBPM was used for the creation of the sample process, Apache ODE may be an excellent choice for deploying on existing servers. Activity is also a good option if the use of Eclipse Indigo is not necessary. The makers of Activity will also probably update their plugin for use with Indigo in the future.

Acknowledgments

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by Space Grant and the National Aeronautics and Space Administration.

Barbara Streiffert – Summer Mentor

Dennis Page – Co-Mentor

Benjamin Smith

Mitchell Schrock

Section 317

_____ _____

Dennis Page Date

_____ _____

Mike Mozingo Date