Planning as a Service: A Plan Repository Model Inspired by Cloud Computing

Khawaja Shams[1], Paul Wolgast[2], David Mittman[3], Tom Soderstrom[4]
NASA Jet Propulsion Laboratory, Pasadena, CA, USA

Arash Aghevli[5], Alfredo Bencomo[6], Ivy Deliz[7],
NASA Ames Research Center, Moffett Field, CA, USA

and

Marc Spicer[8]
NASA Johnson Space Center, Houston, TX, USA

The Ensemble project has consistently identified common patterns pertaining to activity planning across missions, analogs, and field tests. In this process, we have developed a unified planning repository that serves Mars Science Laboratory, International Space Station, Analog Field tests like Pavilion Lake Research Project, and many others. This repository encapsulates complexities associated with persistence, indexing, access and synchronization of data required for distributed planning. By providing a uniform Restful interface to the clients, the plan repository enables access from a variety of clients spanning multiple NASA centers. These clients range from shell scripts and browsers to mobile apps and complex desktop applications. It supports multiple representations, or formats, of plans to support multiple clients. With a focus on interoperability, end-users from disparate clients are able to interact with each other seamlessly. With support for rich querying capabilities through an inverted index of all of the activities and plans stored in the repository, we allow operators and scientists to search for the plans based on tactically relevant criteria with structured and unstructured (full-text) queries. Lastly, in order to support distributed planning, the repository incorporates a messaging layer to synchronize plans across multiple geographically dispersed clients in real-time. In this paper, we describe common patterns in tactical planning and the feature sets we have built to support them. Furthermore, we share key insights and lessons learned from deploying plan repository on a variety of missions and field tests. We provide an overview of the novel integration of technologies that support planning repository, and we discuss the role of cloud computing in our deployment strategy. We conclude with an overview of future research and upcoming features in the plan repository.

[1] Manager, Data Services – Planning and Execution Section, M/S 301-250D. Jet Propulsion Laboratory, California Institute of Technology Jet Propulsion Lab, Pasadena, CA
[2] Member of Section Staff – Planning and Execution Section. Jet Propulsion Laboratory, California Institute of Technology. M/S 301-250D Jet Propulsion Lab, Pasadena, CA
[3] Manager, Software Systems – Planning and Execution Section. Jet Propulsion Laboratory, California Institute of Technology. M/S 301-250D Jet Propulsion Lab, Pasadena, CA
[4] Chief Technology Officer – OCIO. Jet Propulsion Laboratory, California Institute of Technology. M/S 180-300 Jet Propulsion Lab, Pasadena, CA
[5] Software Engineer - Planning and Scheduling Group. SGT / Code TI. NASA Ames Research Center, Moffett Field, CA 94035- Mail stop: 262-4.
[6] Software Engineer - Planning and Scheduling Group. SGT / Code TI. NASA Ames Research Center, Moffett Field, CA 94035- Mail stop: 269-3.
[7] Software Engineer - Department: Code TH. Dell Federal Government. NASA Ames Research Center Moffett Field, CA 94035- Mail stop: 262-4.
[8], NGPS Project Manager. ISS Operations Planner. Johnson Space Center. 2101 NASA Parkway #1. Houston, TX 77058.

I. Introduction

Cloud computing has redefined the development paradigms employed across the industry and catalyzed an unprecedented level of innovation in many diverse areas. Cloud vendors, like Amazon Web Services, Microsoft, and Google are able to inject this productivity into the community by providing the fundamental building blocks required for a diverse set of novel applications spanning multiple industries. Multiple missions across NASA have already started utilizing these building blocks and have deployed applications in the cloud. In November 2010, Mars Exploration Rovers (MER) became the first NASA mission to place a mission critical component in the cloud. Today, all activity plans for MER are stored on Amazon S3 (Simple Storage Service) and indexed in Amazon's SimpleDB. Since 2010, MER has enjoyed 100% availability for their plans, improved the durability, and significantly enhanced the performance of the application. This application previously required a MySQL database, an Apache Server, and a web application server, all maintained in-house. However, Ensemble developers quickly realized that they are able to deliver a transactional persistence layer with hosted services that do not have to be maintained. Meanwhile, MER is also using similar building blocks like Amazon SWF (Simple Workflow Service) to orchestrate data migration, backups, and data processing in the cloud.

While these platforms provide building blocks for numerous applications, the Ensemble project is striving to define and develop Software As A Service (SaaS) capabilities for planning applications. We aim to streamline delivery of planning applications by minimizing overhead associated with design, development, deployment, and maintenance of these applications. We envision enabling missions to dynamically provision components of a sophisticated planning system with the core capabilities available. Specifically, after making specific design choices, missions should be able to get a functional system deployed and ready for customization beyond core capabilities within minutes. While cloud computing enables us to rapidly provision raw infrastructure, we have gone a step further by creating custom capabilities to enhance the provisioned resources with planning software, configure security, isolate instances, and to make the capacity available to the missions. Towards that end, we have preconfigured machine images that provide persistence, indexing, and messaging layers for planning systems. We will continue this development until the entire planning system can be deployed in this fashion.

The NASA Jet Propulsion Laboratory is aggressively investigating the cloud computing options available for missions planning and other applications. With a compelling partnership between the missions and IT, JPL personnel have developed a Cloud Applicability Suitability Model (CASM) that guides selection of a particular cloud based on application specific requirements. JPL currently has production applications deployed in Google App Engine (GAE), Microsoft Azure and Amazon Web Services. For instance, BeAM (Be A Martian), a public outreach website dedicated to exploration of Mars, is currently deployed on Microsoft Azure. We are also investigating hybrid cloud deployments and are currently investigating Eucalyptus, OpenStack, and other options to leverage cloud-based innovations for our in-house infrastructure.

We have validated the integration of our system with the development environment for the Next Generation Planning Systems (NGPS) initiative for the International Space Station (ISS), MSL (Mars Science Laboratory) and MER missions. NGPS is a suite of planning tools being developed as a collaboration between Johnson Space Center (JSC), Ames Research Center (ARC) and the Jet Propulsion Laboratory (JPL) which will address planning needs for both ISS and future Mission Operations Directive (MOD) missions. Score is the planning interface to be used by NASA, the European Space Agency (ESA), and the Japan Aerospace Exploration Agency (JAXA) for authoring the operations schedule and validating it against flight rules and constraints. Score also provides an interface for planning collaboration between remote planners as well as a plugin-based architecture for partners from Marshall Space and Flight Center (MSFC), ESA, and JAXA to contribute their own custom tools.

The Ensemble Project, a highly successful, ongoing collaboration among NASA Centers has supported the development of mission operations software for NASA's Explorations Systems, Science and Space Operations Directorates. Ensemble is designed as an open architecture for the development, integration, and deployment of ground data system mission operations software. Fundamentally, it is an adaptation of the Eclipse Rich Client Platform (RCP), a widespread, stable, and supported framework for component-based application development. The Ensemble Project has been instrumental in the development of NGPS, ISS, MSL and MER operations software.

This current approach to the development of mission operations software has produced a set of powerful tools that have enabled successes for numerous NASA missions. Many parts of the current development process are

functioning well and should be preserved. However, improvements in the state of the art in software engineering and increasing demands from new missions have exposed several areas that deserve attention, including: difficult to test interfaces, lack of user interface design standardization, difficult to integrate software systems, duplication of functionality across tools and overall lack of agility in structuring and delivering new tools. The Ensemble project emphasizes: the use of direct application interfaces over network or file interfaces; a unified cross-platform approach to user interface elements and window management; ease of integration with standard development tooling and infrastructure; and reuse of software components throughout the operations process with a component-based software model.

Ensemble is enabling NASA missions to derive greater results from their investment in mission operations software. Instead of stringing together a series of largely isolated and independent tools, missions are free to assemble precisely the tools they need by drawing components together from different development teams. Mission operators become more efficient, which improves the overall performance of their missions. Finally, operations software developers are free to focus more on developing great tools and less on frustrating integration issues.

II. Core Planning Capabilities

Planning complex systems brings about sophisticated challenges. NASA missions are increasingly becoming more distributed; the Mars Science Laboratory operations team consists of a global community of scientists, operators, and experts. Distributed planning, therefore, is a crucial component in maintaining the collective situational awareness throughout the missions. Within the domain of long term planning, the Ensemble team has had experience supporting operations of numerous missions like Cassini, MER, Phoenix, and ISS. Perhaps the biggest success of the project has been the reuse of code across multiple missions through a modular design of our applications. This paper focuses on server-side components and capabilities that formulate a planning system. By identifying these components, we are able to focus on enhancing components across the multiple missions we concurrently support. This section outlines the common components and the diverse options we have tried across our missions.

A. Plan Persistence Platform

Throughout the years, Ensemble has supported numerous projects with unique persistence requirements. For instance, some projects like ISS require versioning on the persistence layer, while others only desire the latest version of the plan to be stored. On the other hand, while MSL and ISS require full-text search capabilities across the entire plan, MER only asked for the ability to search over metadata (plan name, number of activities, custodians, etc) rather than low level searching over activity names or parameters. We currently support three persistence layers: raw file system, serialized object storage, and Subversion (SVN), a source code version control system.

Plan storage on the file system provides access to the mission operators through a Restful interface, as well as through raw file system access. Privacy and integrity of the data are enforced via POSIX permissions on the file system, and via LDAP integration on the Rest interface. The storage of these plans in a hierarchical file system also affords a natural Restful URI scheme for CRUD (Create, Read, Update, Delete) operations on the plan. To ensure durability of the plan files, traditional file system backup approaches are utilized, and there is no routine health-checks for the integrity and consistency of these files. While this is a simple and elegant solution, there are several problems this approach for a mission with complex planning requirements. First of all, the file system does not automatically provide search capabilities that a database would provide. Hence, an index of the files must be maintained. Although that is true for most approaches, it is increasingly difficult to maintain consistency between the file on the file system and the index. This is because the users are able to go around the server and modify files on the file systems. Without a polling and crawling capabilities built into the server, it is untenable to keep track of such changes on a large-scale file system. Furthermore, the crawling introduces a lag in the consistency between the index and the file system. The second problem with the file system is the lack of versioning support for the files. Once an operator saves a plan, it overwrites the previous copy of the plan. This shortcoming makes it difficult to support auditability and visualization of the changes in the plan over the course of the planning cycle. Furthermore, it makes it nearly impossible to recover from unintentional writes. Despite these shortcomings, this approach works well for several missions that can tolerate eventual consistency in the indexing for out-of-band changes and do not require versioning of the plans that are stored over time. MSL and numerous Analog tests have used this approach to provide a simple, streamlined, and high performing solution to their planners, without too many additional features.

Object-based storage of plans offers Restful access to plans in a highly durable environment with automated integrity checks, and automated replication across multiple data centers. In this approach, plan data are serialized and stored as blobs in a distributed object store and we are able to leverage Amazon S3 as the persistence layer. Once the plan is persisted in the object store as a blob, S3 replicates the files across multiple, physically disjoint, data centers. S3 is designed for 11 9's of durability, which means that if we had a million files stored in S3, we can expect to lose a file once every ten thousand years. In terms of availability, S3 can tolerate concurrent loss of up to two data centers, and still provide access to the data while maintaining durability. While some missions have expressed discomfort in entrusting the availability, integrity, and durability in the hands of a third party entity, MER has enjoyed the benefits, while minimizing risk for this approach. Object-based storage of the plan enables us to persist the data in the cloud in an encrypted format, while never introducing the key into the cloud. This minimizes the level of trust we place on the vendor in terms of the privacy and integrity of the data, while leveraging the research that the cloud vendors have put in place for deploying highly available and durable storage systems. Operationally, this approach also minimizes maintenance overhead that the mission has to deal with: there are no systems or machines that we are responsible for running. This approach shares a few of the shortcomings of the raw file system approach. First, even if the changes to the plan are minimal, the whole plan must be sent over and overwritten. Because S3 currently does not support append semantics, it is impossible to make the system efficient by transmitting small deltas. S3 does, however, offer native support for versioning, giving missions option to automatically store multiple copies of the plan and retrieve historical copies. The second problem with this approach is that it does not provide native indexing of the files. Hence, we have to build the indexing layer on top. However, unlike the native file system approach, all access to S3 goes through Ensemble software. Thus, we are able to update the index whenever we change a plan, minimizing the synchronization issues between the persisted data and the index. This approach has successfully met the persistence requirement for MER, and it offers the lowest operational overhead and minimal cost for running a plan persistence layer. It is also the option with the highest durability and availability.

For missions that have strict versioning requirements and desire to optimize transfers, we have used version control systems like Git[*] and SVN as the persistence layer. Since Ensemble tools are based on Eclipse, we have access to the numerous Eclipse features for version control systems like the views for browsing and comparison, as well as low level tools for the servers and clients to directly interact with the SVN layer for checking out and committing plans. SVN provides multiple options for maintaining consistency between the persistence and indexing layers. We first attempted to use SVN hooks to notify the servers whenever a change is committed. However, due to the lack of reliability of this approach, we have evolved our approach to employ a polling solution that repeated asks the repository for the latest revision. A plan is fully re-indexed upon detection of any changes in a specific plan. This approach requires setup, integration, and monitoring of an Apache server for interaction with SVN or Git for remote clients. The requirement for an additional service makes this approach slightly less desirable, but the native support for versioning, coupled with tooling, make a versioning-based persistence layer very appealing for missions. Currently, the Next Generation Planning System (NGPS) repository for ISS and for several analog field tests is based on this approach.

We have found these three approaches to be sufficient in addressing the requirements for a diverse set of mission that Ensemble sponsors. Furthermore, based on our experience with persistence layers, Ensemble developers are able to quickly analyze requirements for a mission, and recommend a functional persistence layer with an existing codebase. Furthermore, the persistence layer is completely modular in Ensemble's case, which allows missions to cost-effectively change the persistence layer as the requirements mature. In the case of ISS, we quickly migrated from raw file system storage to the version control-based persistence layer, while only affecting the code base for the persistence layer and minimizing the scope of our changes.

B. Search and Indexing Platform

During tactical operations, it is crucial to provide operators and scientists with the ability to quickly search for plans that meet specific criteria. For some missions, these queries are predictable and can be indexed in a database. However, the majority of our missions require deep introspection into plans down to the level of activity parameters.

[*] http://git-scm.com/

For missions with low functionality requirements and a-priori knowledge of the metadata they wish to search upon, we have typically been able to store this information in a database. In the case of MER, we store all the metadata for the plan that the scientists typically search on into Amazon SimpleDB. SimpleDB allows us to delegate the responsibility of maintaining a highly available and highly durable indexing layer to Amazon Web Services team, while requiring only tens of milliseconds in query response time for some of our largest planning databases. SimpleDB also offers conditional puts that allow us to persist plans only if certain criteria are met – thus giving us transactional functionality plan persistence. This approach requires us to store metadata in the cloud in clear text, and it requires the mission to predict the kinds of queries operators would conduct at the time of development of the code. However, it enables our clients to query the metadata and obtain plans without requiring NASA to maintain a single service. The cost is very economical – for MER's plan persistence and querying functionality, the mission ends up spending on the order of a few dollars each month – which is significantly less than running even a single low-end machine. Meanwhile, it gives MER elastic performance that scales with demand during the peak traffic hours.

Most NASA missions require deep full-text searching over all tactically relevant data, including plans, images, and even low level sequences. We extensively leverage Apache Lucene to construct an inverted index of tactical data and to provide fast searching over any of the attributes of the plans. We wrap this capability in a server that can contextualize plans and index relevant portions as they are persisted. The server provides a Restful querying interface that clients, written in any modern language, can interact with and obtain results from. The server supports polling and file system crawling capabilities to detect changes in files after they have been persisted. Today, the Lucene based indexing and searching approach is employed by MSL, ISS, and all analog field tests supported by NGPS.

C. Messaging and Synchronization Layer

Due to the distributed nature of the planning applications, it is important to keep information synchronized between users. Ensemble developers have created a server component to support collaborative editing by maintaining the state of the plans across multiple heterogeneous client applications. Furthermore, a robust messaging layer is also a crucial component in synchronizing state between multiple servers behind a load balancer. Therefore, our generalized messaging layer services both clients and servers. Within Ensemble, we have evaluated XMPP, Redis (a Node.js component), and polling.

We first started with XMPP due to the perceived simplicity. We use OpenFire as our XMPP layer – as it is mature, has a large community, and is fully developed in Java. Our messaging layer treats chat rooms as a per plan collaboration session. To debug the interactions between the clients, a developer can simply use their favorite jabber client to log in to the server and join the chat room. Users can also inject their own messages and watch the client react in a debugger. With XMPP, we experienced several connectivity issues, even for machines on a reliable network. While we can use jabber clients to determine the current members of the chat rooms and diagnose connectivity issues, the code base for XMPP clients is fairly complex. To keep our connections alive over extended periods, we broadcast heartbeats on the chat room. If a client does not hear any heartbeats for a several minutes, it disconnects and automatically rejoins the room. XMPP is engineered for large-scale installations and may be overkill for many of our projects that have membership rates on the order of dozens of users. The planning servers on MSL currently use XMPP to synchronize state with each other. However, the project is eagerly waiting for other projects to come up with new solutions that are simpler and can be plugged into MSL for the next major delivery.

Score, the planning client used for ISS and analog fields tests utilizes a collaboration server to maintain state between all the clients. Score relies on SVN as the persistence layer. Score initially started with XMPP as a messaging layer for the collaboration server as well. Figure 1 shows the original workflow of the system based upon an XMPP implementation. Like MSL, Score developers quickly learned that this is a complex approach for our use case, adds deployment overhead, and has tough low-level debugging. *Score* developers are also actively working on web and mobile interfaces, and the lack of a reliable and native XMPP client in JavaScript made the approach even less appealing. Fortunately, Score is in an earlier phase of development and is able to evaluate alternatives that can be directly fed back to other users of the messaging layer.

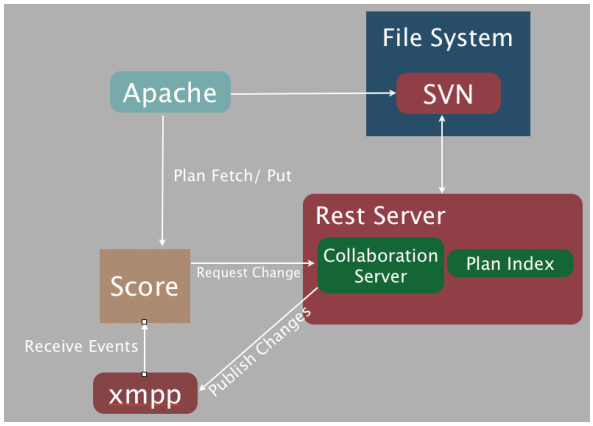

Figure 1. The ISS Planning System High Level View (XMPP variant)

We have prototyped a new messaging layer based on Redis as the message broker. Redis is a lightweight, highly performing, key-value store, with extensions to support queues and publish subscribe mechanisms. This approach is appealing for many reasons. First, we are able to connect clients in many different languages directly to Redis. Specifically, we use jedis to connect our java clients and our java servers to the Redis server and to subscribe to channels for messages. To support the JavaScript clients, we have a simple node.js script that connects JavaScript clients to the Redis channels through socket.io. Figure 2 provides an overview of this architecture. While it offers a lightweight and simple solution that supports all of our clients, Redis deployment requires us to open up two additional ports on our servers: one for Redis, and one for the Node.js server. We are currently investigating ways to simplify the deployment in a production environment.

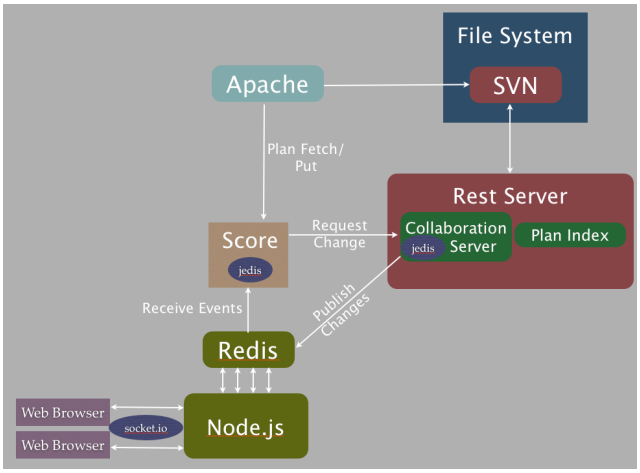

Figure 2. The ISS Planning System High Level View (Node.js/Redis variant)

The simplest solution relies on polling the collaboration server for changes. This approach obviates external servers and extra ports at the cost of latency and additional development on the clients to support the polling. It supports browser clients as well as clients written in any modern language. Furthermore, it minimizes concerns about clients disconnecting under unreliable networking conditions and dropping messages. To minimize latency concerns, we are currently working on a solution that augments the polling clients with a real time notification scheme based on the Redis publish-subscribe mechanism. In this hybrid approach, the clients would poll on a regular interval, but if they receive a message, they will poll immediately.

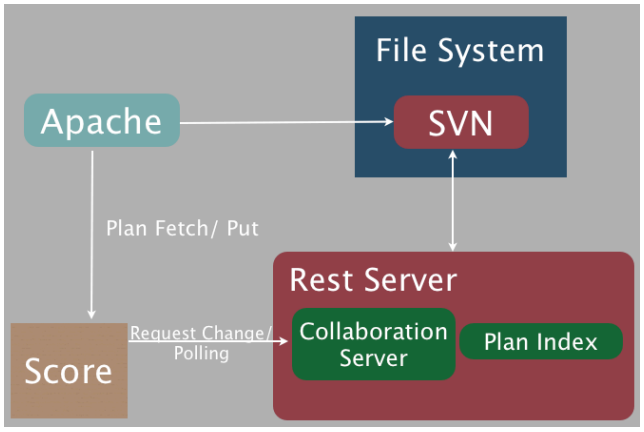

Figure 3. The ISS Planning System High Level View (Polling variant)

III. Planning Components as A Service

D. Virtual Machine Images

Leveraging advances in virtualization, cloud computing provides tools that significantly streamline the deployment process for applications. One of these great tools is virtual machine images – or AMI (Amazon Machine Images) as they are typically known in popular public and private cloud environments. In a cloud environment, developers can configure a machine, install necessary software, and create a snapshot of the fully configured machine into an AMI. An AMI can be instantiated on-demand into as many machines as desired, delivering a seamless process to replicate production, staging, and development environments.

As part of our research and development, we have used cloud resources extensively to support the multiple development machines needed to build our products and test them. Within these research environments, we generated a set of base images that contain the planning components described above. Our catalog of images include pre-configured images for persistence, messaging, and search indexing layers – and they streamline the process for an Ensemble project to test a new approach for one of our key components. An image is not tied to a particular physical machine or the size of the machine. Once an image is created, a project can chose to run it on a larger machine based on real-time requirements. Furthermore, for project that are employing load balancing capabilities, these images also facilitate horizontal scaling by allowing projects to instantiate as many copies of the image as needed.

E. Automated Build Deployment Services

Ensemble employs continuous integration as a core technique in agile software development. Given that Ensemble's source repository is shared with several hundred developers and dozens of products, it is crucial for us to exercise any committed changes against any product they affect. Unit tests are encouraged in the ecosystem, as they are a great tool in detecting when a commit violates any assumptions developers may have made previously or if a bug has been introduced. Towards that end, we extensively utilize Atlassian Bamboo to automate software builds, unit test execution and in some cases, deployments to production environments.

Within Bamboo, we have created a deployment scheme for our projects that wish to develop with the latest development of the server. For instance, whenever any source file impacting our server for the ISS or analog tests are committed into our code repository, Bamboo kicks of a build, runs unit tests, and if the unit tests pass, it deploys the latest build into a development server environment.

We are currently working on a deployment scheme that runs multiple builds of the server concurrently to enable efficient debugging if the developers experience a regression in the code. In this scheme, we would have a stable server with code that has been graduated after proving maturity in the development environment, as well as a development server that is fully synced with the repository's latest commits. We are also working on a web interface that will allow developers to deploy a particular version of the server on a new machine. Upon a developer's request,

we would provision a new machine, push our server on it, configure the security groups, and deploy the server, and provide the developer with the IP address of the machine. We believe this capability will enable client-side and server-side developers to quickly integrate with working copies of the server, and it will also help us push out production servers to be used in analog field tests.

IV. Conclusion

This paper covers the component-driven development scheme within Ensemble for Planning Software that allows us to provide "Planning as a Service" capabilities to our projects. New missions using Ensemble can construct their planning system with components that best match their requirements, and they can rapidly prototype their systems through Ensemble's collection of AMIs as well as complementary code for each approach. This capability creates an ecosystem where components can continuously evolve independently across projects, with innovation in each area flowing across the missions. Our extensive use of cloud computing enables us to deploy our system in real-time, and it significantly reduces the costs associated with prototyping a new capability. We are quickly moving towards the provisioning of an entire planning system, from tactical downlink data processing, indexing, data delivery, planning, and synchronization with a click of a few buttons.

Acknowledgments

We would like to thank the SPIFe team at NASA Ames Research Center for their tireless development efforts towards the research described in this paper. Specifically, we thank Melisa Ludowise, Michael McCurdy, Guy Pyrzak, and Sam Hashemi. We would also like to thank members of the Operations Planning Software Lab, including Megan Mickelson and Tony Valderama for their contributions in our research. Part of this research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

[1] Michael Armbrust et al. Above the Clouds: A Berkeley View of Cloud computing. Technical Report No. UCB/EECS-2009- 28, University of California at Berkley, USA, Feb. 10, 2009

[2] Khawaja S Shams, Dr. Mark W. Powell, Dr. Jeffrey S. Norris, Tom Crockett, Tom Soderstrom. "Cloud Sourcing Cycles: How Cloud Computing is Revolutionizing NASA Mission Operations," AIAA Spaceops 2010.

[3] Mark Powell, Thomas M. Crockett, Jason M. Fox, Joseph Joswig, Jeffrey S. Norris, Khawaja Shams, Recaredo Jay Torres, "Delivering Images for Mars Rover Science Planning," IEEE Aerospace 2008.

[4] Eclipse Equinox (2010 February). *Equinox.* Available: http://www.eclipse.org/equinox/

[5] AWS (2012 May). Amazon Web Services. Available: http://aws.amazon.com

[6] AWS S3 (2012 April). Amazon Simple Storage Service. Available: http://aws.amazon.com/s3

[7] AWS SQS (2012 March). Amazon Simple Queuing Service. Available: http://aws.amazon.com/sqs/

[8] AWS SWF (2012 May). Amazon Simple Workflow Service. Available: http://aws.amazon.com/swf/

[9] The Ensemble Project, "The Ensemble Canon," NASA SP-2011-597, 2011.