User-Driven Collaboration for NASA Mission Control

Christopher Webster  
SGT, Inc. / NASA Ames Research Center  
chris.webster@nasa.gov

Nija Shi  
SGT, Inc. / NASA Ames Research Center  
nija.shi@nasa.gov

Sue Blumenberg  
SGT, Inc. / NASA Ames Research Center  
sue.blumenberg@nasa.gov

Irene Smith  
SGT, Inc. / NASA Ames Research Center  
irene.s.smith@nasa.gov

Sylvia Lin  
UARC / NASA Ames Research Center  
sylvia.lin@nasa.gov

Benson Hong  
UARC / NASA Ames Research Center  
benson.hong@nasa.gov

Adam Crume  
UARC / NASA Ames Research Center  
adam.a.crume@nasa.gov

Peter Tran  
NASA Ames Research Center  
peter.b.tran@nasa.gov

Abstract

NASA Ames, in conjunction with the Johnson Space Center (JSC), is building a platform to enable mission control operations software to be assembled from flexible collections of components and services. MCT is designed to support rapid creation, composition, visualization, and certification of user objects, while using social media techniques to help communicate, categorize, and analyze the massive amount of data and complex processes required for space flight operations.

Since its initial deployment, MCT has helped spur collaboration within and across NASA centers, allowing different teams to work together on the shared goal of safe spaceflight. Flight controllers at JSC are assembling, tagging, searching, and visualizing telemetry. Software developers are extending the platform by integrating new data sources and analysis capabilities while reusing high-level functionality, such as scalable real-time plotting. Finally, the process of adopting MCT across NASA centers has opened communication channels through shared software development, feature design, and work practices.

1. Background: Mission Control Overview

NASA’s Mission Control Center (MCC) at JSC operates all Space Shuttle missions and all activity onboard the International Space Station (ISS). ISS control requires vast amounts of data to be displayed on each monitor, continuously updated at high rates, for years—much higher requirements than exist for any other spacecraft. ISS flight controllers currently rely on over a thousand separate software applications to perform specific monitoring and control functions. These applications were not designed to work together and cannot be easily modified to meet changing operational needs. New requirements require extensive modifications, additional applications, or in some cases, cumbersome workarounds. MCT allows components to be assembled and modified by users, with the level of composability and permissions controlled by MCT policies. By assembling systems from components using a common framework, the traditional “walls” of monolithic applications are eliminated, as are the heterogeneous code bases and functional overlaps that come with traditional software applications.

MCT [11, 12, 13] is organized around user objects that are designed to mirror the data and behavior of their real-world counterparts in the user’s domain. Some examples of user objects are telemetry elements, procedures, commands, physical parts such as pumps or batteries, and collections of other MCT objects. MCT displays are compositions of views for these user objects. Traditional application boundaries do not exist; user objects from multiple domains may be composed as needed. User composition empowers users to make rapid changes to visualizations, within the constraints of organizational specified polices, without the need for code changes. Distributed object sharing supports synchronization of user objects and views across multiple workstations in a clustered MCT deployment. MCT is the foundation for the next-generation of user tools in JSC’s Mission Control...
Center and potentially for other space mission operation centers, both public and private, for spacecraft having telemetry data ranging from the size of the ISS to small satellites.

2. MCT Architecture and Design

MCT was designed by flight controllers for flight controllers to safely operate space missions. A front room flight controller at JSC needs to monitor upwards of 5,000 pieces of telemetry for each discipline, aided by other supporting flight controllers in the back room. The typical flight controller will use a two-dimensional plot to monitor data that needs trending information (for example, pressures and temperatures), alphanumeric displays (as well as evaluations such as enumerations) in the form of tables to visualize discrete data, and a configurable rule-based system to help detect anomalies. Flight controllers are trained extensively to recognize patterns, so the ability to quickly scan and visually recognize deviations in densely packed displays spread over multiple monitors is essential. These displays need to be usable not only by flight controllers from the same discipline but also composed into larger overview displays that may be used by another discipline. For example, the flight director will monitor overview displays created by each discipline.

The MCT platform is reusable, as it provides a software module and windowing system, but it is not too generic, as the platform is usable for command and control operations as delivered. The Eclipse or NetBeans Rich Client Platform (RCP) could have been used as the basis for the platform, and indeed it was in early prototypes; however, this approach was ultimately rejected in favor of using the Eclipse module system and creating a visual metaphor targeting the space mission operations community. The MCT platform is designed for immediate use, as the default platform ships with a set of modules including table and plot visualizations, telemetry definitions, evaluators, and the feed aggregator system to connect with existing telemetry data streams. Currently, there are no widely adopted standards for ground-based systems to acquire telemetry, so integrating the telemetry stream is the only requirement on the platform user. However, given the widely diverse analysis requirements per mission, additional view development may also be required.

A generic RCP-based approach is most commonly structured on source control or another hierarchy directly supporting the underlying code structure. In contrast, the objects in the space operations domain are not normally recognized as a hierarchy; in fact, the object taxonomy needs to support many different hierarchies and non-hierarchical relationships (e.g., even in an apparent tree such as in the figures in this paper, some objects appear indented under more than one parent) that are known only after the system is deployed. The ISS has more than 200,000 pieces of telemetry that change based on the current events on board (for example, payloads and which astronauts are on board). Thus the natural way to discover telemetry is to use search, and both hierarchical and non-hierarchical structures to represent the relationships between object compositions and design the user interface around these relationships. However, the most important factor when rejecting the RCP was the control and rendering performance of the graphical libraries. The NetBeans windowing system did not provide sufficient control capabilities and the Eclipse SWT performance on Linux was problematic on the large number of plots required to support real-time flight operations.

The client architecture incorporates OSGi (Open Service Gateway Initiative) [4], which is used in the Eclipse Integrated Development Environment (IDE) [5] to provide classpath isolation (the ability to have a unique classpath for each bundle that prevents conflicting class versions from interacting across bundles) and version support (to validate that dependencies are satisfied at the Java package level), as well as a service model. In addition to the capabilities described above, the service programming model isolates faults in specific code bundles that can be handled generically in the platform and thus restricts errors to specific functionality instead of risking spread to the entire application. Finally, OSGi provides the ability to dynamically deploy software bundles. This capability allows a patch to be applied to a running system and could be used to support emergency time-sensitive bug fixing.

The MCT client was developed in Java to support the intensive graphics requirements of plotting in the control center. Any one of the many ISS displays can show up to 5,000 pieces of telemetry or ground data. On average, each is updated once per second, but some update 100 times per second. In our investigation, we found that 2D drawing (such as that required by plotting) was more than two orders of magnitude slower (in the best case) when using JavaScript than when using Java. The investigation tested multiple JavaScript plotting packages in Chrome, Safari, and Firefox. We are currently investigating how to provide filtered access to displays outside of the control room (including both mobile devices and web browsers). This proves a challenge for several reasons including security (specifically around the data values) and work practices that were not designed with sporadic connectivity in mind. For example, what happens if the
vehicle must be controlled remotely and connectivity is sporadic?

The MCT platform uses a relational database as the internal storage mechanism. The database contains information about the user objects and not telemetry values, which is similar to how a mash up editor would store information on the mash up configuration but not the interior of the views. MCT serves to combine multiple disparate data sources (for example during spacecraft operations this would be the telemetry metadata as well as real-time and archive data stores for the telemetry values). Database replication provides read availability during failure of the master (write) instance. The database stores the component models and the relationships between components, as well as the persistent view information. This information is currently made available only through direct database calls, but there is work planned to provide a web service interface to support alternate clients. For example, a web or mobile client could provide alternate visualizations for MCT components.

The ability for both clients and servers to continue execution when services are unavailable is essential, as are bounded CPU and memory usage in order to support cohabitation with other applications. In addition, there are requirements specifying the time between receiving a telemetry value and when it is displayed to the user. These requirements are similar to Internet applications today; however, the difference between MCT and a web site is the rigorous certification process (described below) required before software can run in the MCC. The certification process serves to slow the rate of change to the deployed software and requires the platform to handle software faults while preserving operations. These requirements can largely be met by redundancy and failover processes at the hardware level where a machine has redundant components (e.g., multiple disks) as well as at the software level (e.g., the database employs replication).

The MCT platform provides services such as policy management for domain experts to control aspects of the platform. The policy system allows control over the CRUD (create, read, update, and delete) aspects of the system. The policy system can also control which views are available on which component types, view ordering, and composition (the embedding of components into other components). This is currently done in Java code; however, work is underway to investigate adding support for both declarative policies and user creation of policies using a visual policy manager.

3. Collaborative Monitoring and Control Features

The MCT platform includes features that support collaborative spacecraft monitoring, command, and control. MCT also facilitates JSC’s MCC certification process.

3.1. Composable Components

The MCT platform defines a component model based on the model-view-controller (MVC) pattern. A developer can provide arbitrary component types, views that can be attached to those (or any other) components, policies that allow dynamic control over behavior such as visibility of views, menu items, component creation handlers that are notified when a new component is created, and status widgets that are displayed in the window’s status bar. The platform also provides additional services around telemetry management, such as a feed aggregation service that supports the ability to incorporate data feeds from multiple sources such as live, predictive, and archival sources.

MCT is designed around user object composition, which allows flight controllers to easily create complex displays. A user object composition is created by populating a collection component with related components. The composition can be configured using the appropriate view. A view type (such as plot or canvas) is created by a software developer while its applicability to a component is controlled by policy. A view is configured by users with the appropriate permissions. The views range from analysis views such as plotting and tables to configuration (editing properties about the component such as the name).

3.2. Certification Workflow

Certification, in the context of this paper, refers to the rigorous review process that a display (specifically alphanumeric) must undergo before it can be used to base decisions on during a mission. The certification workflow is a people-process independent of the MCT platform. Prior to MCT, a flight controller (or perhaps an administrator) would need to make hand edits to files that mapped labels to telemetry elements in terms of pixel coordinates. This process is manual, tedious, expensive, and error prone, and thus requires verification to ensure the labeling has been applied correctly. Another side effect is that making changes during a mission (for example, to look at a new piece of telemetry) can be too time-consuming when time is of the essence.
With MCT, a piece of telemetry can be viewed as an object, and a display can be viewed as a collection of references to these objects. The same telemetry object can be referenced in many display collections. Thus, a change in a telemetry object is reflected in all display collections referencing this object. This not only speeds up the process of labeling but provides additional safety as the mapping is only required in a single place rather than for each display.

Verification of new labels can be accomplished collaboratively as well as automatically in MCT. Objects are shared across the cluster, so a change to an object in one instance will eventually update all instances. MCT takes advantage of this cluster environment by providing a set of shared collection objects acting as communication channels to support a collaborative user certification workflow. Those collections can be automated to support the workflow. For example, when a user creates an object and wants to have it approved for viewing by other flight controllers, the user drops that heretofore private object into a collection named “Ready for Peer Review.” That collection is visible by other members of the user’s flight control discipline. The action of dropping an object into that particular collection automatically checks that object’s property “Ready for Peer Review,” and makes that object read-only except for the other property “Ready for Approval.” When any other user decides that object is okay for approval by the flight discipline’s leader, the other user marks the object as “Ready for Approval.” That automatically removes that object from the “Ready for Peer Review” collection, adds it to the “Ready for Approval” collection, and makes the object read-only by all users except the leader. Alternatively, the peer-reviewing user can drop the object into the “Ready for Approval” collection, which automatically checks that object’s “Ready for Approval” property and removes the object from the “Ready for Peer Review” collection. Regardless of which mechanism the peer reviewer uses, the result is that the leader can edit one property of the object—“Ready for Certification”—or equivalently drop the object into the collection named “Ready for Certification.” The process continues up more levels of workers until the object is in a “Certified” collection that is either visible to only the members of that discipline, or to members of all disciplines. The certification workflow is clearly defined and controlled by the policy manager to avoid human errors. This could be enhanced further to support other notification mechanisms such as email.

3.3 Customizable Labeling Algorithm

With thousands of labels and values associated with constructing large dense displays by flight controllers, there needs to be an easy and intuitive way to abbreviate and shorten labels while ensuring consistency in the computation of the abbreviation. MCT has a user-customizable labeling algorithm as part of its platform for plot legends and tabular alpha display views. In applications before MCT, labeling was handled in each display by requiring the display author to type in a label for a piece of telemetry. This was problematic in several ways: The only safety guarantee that each abbreviated label was consistent with the telemetry’s official name was review by other flight controllers, which was expensive, time consuming, and error prone. The labeling was inconsistent, as each discipline could use its own naming convention, so displays created by that discipline were not easily understandable to people outside that discipline. MCT’s labeling algorithm provides better safety, because it guarantees that all the pieces of the telemetry’s official name are visible somewhere near that telemetry’s value—adjacent to the value, or part of it in a column or row header or panel or window title. MCT’s labeling is automated, eliminating manual verification of each label, resulting in predictable abbreviations. It supports composing views without extensive editing, as there is naming consistency around the underlying telemetry values that can be enforced by software instead of by work practice, thus making collaboration with shared compositions possible.

For example, suppose a flight controller wants to compose four rather lengthy telemetry elements in a 2x4 tabular alpha display view or as a single column with four rows within a plot legend display. The telemetry elements’ official names are as follows:

- PFCS_1B_Pump_A_On_Off_Discrete_Status
- PFCS_1B_Pump_B_On_Off_Discrete_Status
- PFCS_1B_Pump_A_Speed
- PFCS_1B_Pump_B_Speed

Figure 1 shows how these telemetry elements would be displayed in a plot view (some of the full names are truncated and the underscores are automatically replaced by white spaces). Figure 2 shows how they would be displayed in a table.
These labels are quite cumbersome, so the flight controller would like to apply the customizable labeling algorithm to truncate the keywords “PFCS 1B” and “Off Status” from the plot legends to save space, or automatically re-arrange the tabular alpha view to eliminate the common keyword context between “Pump,” “Speed,” and “Status” to shorten the table display. The labeling algorithm is triggered by the user adding the keywords to the panel and canvas title within MCT views. The labeling algorithm ensures the full textual label for a value is always visible even if its pieces are displayed separately. The label adjacent to the value can be shortened, but the combination of window title, panel on canvas tile, and row and column labels must always produce the original name. This requirement ensures that mislabeling cannot occur as a result of human error.

After applying the labeling algorithm for plots, the keywords “PFCS 1B” and “Off Status” are stripped out from the plot legends, as shown in Figure 3. Note that all stripped-out words are now added to the plot or table title, so no information is lost.

Figure 1. Initial plot legend labels

Figure 2. Initial 2x4 tabular alpha labels

Figure 3. Shortened plot legend labels

Figure 4. Shortened 2x4 tabular alpha labels

3.4 Activity Separation

“Activity separation” is a JSC term for a requirement that can go by different names in other spaceflight centers: for example, separation of simulated from real spaceflight operations activities—the spaceflight data and the monitoring, decisions, and commanding activities based on those data. It is dangerous for flight controllers who nominally are acting on live data to instead see even one piece of simulated data or for flight controllers who nominally are issuing simulated commands to leak through to the live system and affect the real vehicle. Separation is needed not only for simulated versus live activities, but also for one vehicle versus another, for one kind of activity versus another (e.g., experiments on vehicle versus crew time off), and for levels of safety (e.g., dangerous docking that requires multiple levels of approvals of decisions, vs. crew sleep). This is similar to software development’s need for test, staging, and production environments.

MCT strongly supports activity separation; without it, MCT would not be usable for spaceflight operations at JSC or anywhere else. An example is the ability of a user to create an object/display in a
simulation activity, and then use it in another activity such as live operations via a certification workflow. MCT was designed specifically to support not only the previously described collapsing of data and application boundaries within an activity, but also to keep each activity separate, and to ease the use of objects and views across activities in tightly policy-controlled ways. MCT was designed to support all of these requirements using either logical or physical separation of the activity database and easy migration across databases with whatever degree of automation is desired by safety policies. MCT thereby supports collaboration across activities.

4. MCT User and Community Collaboration

MCT’s end users are flight controllers. The MCC is organized into disciplines such as power, trajectory, and data communications. Each discipline is composed of “front room” (the main mission control center) and “back room” (support rooms) flight controllers. Front room controllers monitor and control systems in real time and are the decision makers, while back room controllers are engineers who are trained in their specialties and are able to more extensive problem analysis of problems to aid the front room controller in the decision process. When an anomaly is discovered by a front room flight controller, support is requested from the back room engineer to analyze the problem in the global sense: across flights, between systems so the root of the problem can be understood.

4.1. User-to User-Collaboration

Social productivity refers to the power of the crowd to improve the user experience. As more users congregate on a site, additional content is produced. The more content is produced, the more users come to the site and the cycle repeats. The best example of this phenomenon is Wikipedia, where the entire Internet community works together to codify human knowledge. The following sections show how MCT incorporates this technique to help categorize the massive amount of telemetry available from the ISS as well as aiding real-time problem solving between front and back room flight controllers.

4.1.1. Object Sharing. Before MCT, end users could not modify most JSC telemetry displays, and those that could be modified were modifiable in only superficial ways. Flight controllers were required to submit modification requests to NASA programmers (typically contractors) specialized in each tool, incurring time and cost for the MCC. Now, flight controllers can modify, compose and (as described in other sections) certify displays.

Flight controllers can combine telemetry either for their own private use in a sandbox, through the certification workflow described previously (that can be made available as a standard set of components for the discipline that are visible within or across disciplines based on the visibility policies), or shared directly with any other MCT user if policies permit. Direct sharing makes use of a concept called a drop box and involves dragging a set of components into the drop box. The drag gesture makes the component available to the user (or set of users, depending on the drop target) for viewing or modifying based on policy, and also changes the state of the object to the shared state.

Once a component becomes shared, modifications to the state of the component (including the persistent state of its views) require coordination to prevent issues like lost updates. MCT currently supports the ability to unlock an object for exclusive writing (pessimistic locking) where the object is not available for editing until the lock is released. The common work practice is to do most development in a user’s sandbox prior to sharing objects (specifically for building displays that are destined for certification). However, MCT also provides facilities for manually unlocking objects (for the case of a locked workstation) and automatic detection of stale locks (where a workstation crashed while holding a lock). Locking enables atomic changes, thus, the entire set of changes is either persisted or reverted, depending on the user’s desire prior to releasing the lock. Committing the changes will cause all references throughout the cluster to be updated.

In addition to locking, MCT supports “twiddling”—the ability to modify objects only locally, with no risk of exposing the modification to anyone else. Twiddling is possible for all objects and does not require exclusive write access to the underlying component, as the changes will only be applied locally (essentially, a copy of the component is created and all changes are applied to the copy). A future release will support the ability for the user to persist changes, if no other changes have been made to the component or its views at the same time. If changes have occurred, the user will have the option to abandon the changes or to create a new private version of the component.

4.1.2. Tagging. Tags [7] allow dynamic classification of data by the user community. In the context of NASA, scientists, engineers, and flight controllers are confronted with massive amounts of telemetry that represent numerous functional areas and scientific
disciplines. Tagging allows discipline experts to apply their view of the data without requiring additional development. As more components are tagged, common terms can be extracted and used to define dynamic shared vocabularies, creating a folksonomy [7, 8]. However, niche tags can also be created to support any domain without requiring an upfront agreement as in other shared vocabularies. The ability to rapidly associate user-created components with tags is an important capability when encountering unknown situations, as tags to group components can instantly be created for immediate action or for later analysis.

The first uses of tagging at JSC were to attribute telemetry objects with a specific version of recon (recon is short for “reconfiguration” and is the set of telemetry that is currently in the vehicle). A new recon set is created to accommodate changes to the vehicle as well as the phase of flight, and so is one way in which the previously described “activity separation” is defined. This tagging allows users to quickly identify which telemetry was valid for which recon set. Tagging has also been employed to support the certification workflow described previously. Tags are used to identify the state of components (ready for review, peer review, change required, certified) and are updated to reflect the state of the process. Additional tagging patterns are also likely to emerge around a variety of criteria such as discipline identification and logical subsystem as adoption increases.

MCT’s tagging implementation uses relational database technology, which provides the usual database safeguards such as referential integrity and high-performing search keys. Tagging is currently used to attribute telemetry to specific missions as well as specify components which have been vetted in the certification process. Exploratory work is currently underway to provide an alternate navigation view based on search and tags. The view supports the ability to work with tags using search as well as navigate using tag clouds.

4.1.3. Chat. A plugin was developed to incorporate XMPP [6] into MCT, allowing MCT users to collaborate through instant messaging (Figure 5). Each message that is exchanged can be recorded for traceability as well as safety. In addition to exchanging textual messages, URL-based references to MCT components can be exchanged. Clicking on the URL opens the object into a new window, making exchanging component references easy (Figure 6).

4.1.4. Collaborative Workflow. MCT supports collaboration between component authors and component certification reviewers through MCT’s certification workflow features. There are communication challenges when ensuring a display meets the requirements of a specific flight controller discipline, including verifying the label attribution correctness and discovering the relationships between telemetry elements. Before MCT, component certification relied on external process to initiate the review process. With MCT, component certification workflow can be automated and standardized using various MCT features, including automatic extraction of semantic relationships and tagging.

As described in the architecture section, telemetry displays are customizations of collections. The references between containing and contained components provide a mechanism for extracting the referencing relationship, as these relationships are retained as part of the component data model. This is
similar to techniques in software development environments for refactoring (renaming across all references, for example). As the flight controller community builds up the repository of compositions, the network of relationships between components continues to grow. This property is a concept similar to adding friends in Facebook, where the human social connectivity graph is being constructed one friend at a time. Flight controllers are creating the vehicle ontology through the visualization compositions that are created. This can even include friend-like relationships by adding users to a collection.

The relationships can be exploited to visualize additional relationships such as what compositions (and hence which displays) reference specific telemetry or to find all compositions that reference a specific collection. Providing the ability to quickly view references can aid flight controllers when examining failures, as references to a failing component that could cascade can easily be shown. There are many possible visualizations for this information, but an experimental capability in MCT shows these relationships as a force-directed layout that can show all incoming references. These references can be expanded until the root component of MCT is encountered, which allows the analysis scope to be easily expanded.

![Figure 7. Ancestor View](image)

4.2. Building a Community – Developer-to-Developer Collaboration

This paper has thus far considered only the user community. The user community will discover problems, suggest enhancements, and request new functionality not envisioned when the software was originally developed. Thus in addition to fostering an active user community, providing a platform that can be extended by developers outside the original team can increase the overall usefulness of software.

One metric for the value of an open source community is the activity of the community. MCT is no different and is building up the community in several ways: providing usable mission-hardened software to the space operations community, participating in conferences and calls for papers within and outside NASA to increase awareness, and reaching out to mission control centers that are not using MCT to learn about the operational requirements to determine what capabilities should be provided in the platform. The efforts to date have garnered interest from multiple NASA centers as well as other mission agencies such as universities. The current governance model is that Ames leads the platform development effort and is expecting to add code stewardship based on adoption and contributions.

4.2.1 What Developers have Created. The MCT platform is deployed with a set of components, services, and views. The components include collections and evaluators. Collections provide a container for an arbitrary set of components; the most common scenario is to group related telemetry into a collection. Evaluators provide a way to interpret an extensible set of evaluation languages based on specific feed inputs. The MCC deployment of MCT provides enumerations where integer telemetry values are matched against specific values (hash table), discrete (where a non-zero value outputs a specific message and a zero value outputs another) and multiple discrete (where a group of non-zero values must be considered as a group, for example a specific message should be provided if more than one value is non-zero).

A development team at JSC has developed a viewer for procedures (a documented set of steps that requires some manual steps, typically telemetry value validation). This component incorporates the plot and alphanumeric values as part of the procedure view to reduce the manual effort required in validating telemetry values.

MCT built-in services include a data feed service, policy management, and event log service. When MCT is deployed to a new environment, site-specific services are integrated (for example, adding a telemetry stream using the data feed API) by the addition of OSGi bundles. MCT can easily incorporate web services to support telemetry metadata and value retrieval (telemetry as a service), issuing instructions to the vehicle (commanding as a service), and user and role services. This provides an easier integration model than writing glue code and can reduce the time to implement ground-based systems for missions. This is an area where standardization of telemetry interfaces could aid in the reuse of telemetry acquisition code.

The platform also provides a set of core views for monitoring live and predicted telemetry in the form of alphanumeric tabular and two-dimensional plotting
views. A canvas view is also provided to support free-form layout of views on a two-dimensional surface including tiling, z-ordering, and snap-to-grid features.

4.2.2. How Developers are Collaborating. One only needs to look at development communities around platforms such as Facebook [9] to see that entire companies, such as Zynga [10], a social gaming company, have been formed to capitalize on the strength of the platform. While MCT is currently targeting a community much smaller than Facebook, being able to reuse common rich and optimized visualizations (for example, plots and alphanumeric displays), can save time and ultimately money while increasing safety.

Collaboration in developer communities revolves around code. MCT was developed around a platform, so the views and components made available from the MCT development team use the same APIs that are available for any developer. This aids in discovering functional or usability gaps when developing components and views.

MCT is considered GOTS (Government Off-The-Shelf software) and thus can be used across different NASA missions and across different government agencies. As mentioned in the architecture section, MCT uses OSGi to support adding or removing functionality by configuring the set of modules that are available to MCT. This can be done simply by adding or removing bundles (JAR files with OSGi-specific metadata) from a specific location in an MCT deployment. The ease with which functionality can be added into MCT allows components, views, and other functionality to be made available for assembly by specific missions, similar to what happens in an app store when a new piece of functionality is added to a phone. Functionality can be described by features as well as maturity so individual missions can determine which capabilities to configure.

NASA’s Jet Propulsion Laboratory (JPL) is currently investigating MCT. JPL was attracted to the platform by the platform capability, specifically the plot and table views. A telemetry data feed plugin has been written to integrate JPL telemetry as a set of RESTful [14] resources.

One often overlooked aspect of collaborative technologies is the ability to foster a vibrant developer community that can help decrease the time to launch a new mission by providing a marketplace for functionality [1] (similar to Apple’s App Store [2] or the Android Marketplace [3]). The MCT marketplace, shown in Figure 8, contains functionality and ideas that can be incorporated into a deployment.

9. Conclusion

MCT serves as the Rich Client Platform for space mission operations. The platform was developed around social networking inspired features like chat, tagging, and crowd sourcing. The mission operations community requires features like activity separation and certification to ensure safe successful missions. However, MCT also incorporates open source community development techniques (like open architecture and APIs) that harness the developer community to collaboratively enhance the software. The collaboration features enable flight controllers to work together to analyze the massive amount of telemetry data available from ISS. The developer platform allows MCT to be extended in ways that were not envisioned in initial deployments. Collaboration in both the user and developer communities enables more effective mission operations now and into the future.

10. References

[10] Zynga (http://www.zynga.com/)


