# Reinventing User Applications for Mission Control

Jay Trimble
*NASA Ames Research Center, Mountain View CA, 94035, USA*

Alan Crocker
*NASA Johnson Space Center, Houston TX, 77062, USA*

## I. Introduction

*Scenario, Present Day* – In a simulation de-brief, flight controllers discover the need to build a multi-discipline composite display. They submit a software change request to platform services. Constrained by a traditional software change process, the changes roll out and are available for use months later.

*Scenario 2011* – In a simulation, flight controllers discover the need to build a multi-discipline composite display. After the de-brief is over, one flight controller stays an extra hour. Using Mission Control Technologies (MCT) certified composable user objects, the controller builds the new displays. Within the hour, the display is ready for use.

*Exploration Scenario* – A future mission control center has continued what the International Space Station Flight Controllers have begun—the combining of what were previously multiple flight control positions into one. A slimmed down mission control center is guiding a descent to the lunar surface. Using an MCT multi-disciplinary display composition, a flight controller executes a procedure. Within each procedure step, commands and telemetry are composed, so no context switching between software packages is required. On a nearby console, another operator is viewing some of the same information. She has composed only the telemetry points she needs, and she's viewing them as plots, not alphanumerics, as in the procedure steps.

These scenarios illustrate the capability of technology now being installed and tested in NASA Johnson Space Centers (JSC) Mission Control Center (MCC). MCT is developed by NASA Ames Research Center (ARC), in collaboration with JSC. Key MCT concepts are:

- Users are provided with composable software components (called *user objects*) which they may compose without the need for change requests or other platform service support.
- "Walls" between traditional software applications are eliminated, allowing for operations that are unconstrained by traditional application barriers and stovepipes.
- Policy control for managers enable and limit flexibility, as per organizational policy.
- Certified software components for platform service providers allowi them to provide users with secure software that the users can configure themselves.

What is MCT?" We'll give an overview here then spend much of this paper answering that question in more detail.

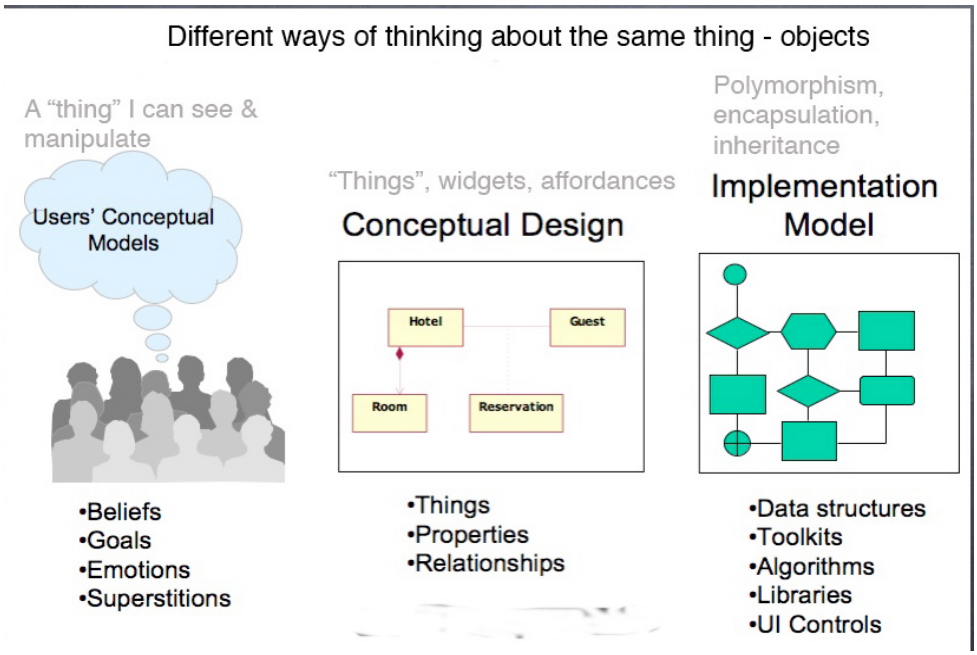



**Figure 1. Users, designers, and developers perceive software differently (modified, original courtesy of IBM)**

### A. Different Perspectives

As we talk about software, it's important to remember that users, designers, and developers see things from different perspectives. Where a user sees and interacts with software only through the user interface, a developer sees code, objects and toolkits. Let's take a simple example—an object. Mention the word "object" to a developer and they are likely thinking about class diagrams, inheritance, and polymorphism. A user, on the other hand, simply sees things on the screen. They can interact with those things through the user interface, but they do not typically perceive those things through the underlying code. A designer understands the intent of the user interface design and interactions, whereas a user must discover how the software works through direct interaction.

MCT is:

**For Users**


From a user's perspective, MCT is a collection of composable user objects, which we sometimes refer to as "things." A user object is a functional piece of software, such as a telemetry value, a procedure step, a timeline, or an activity. It is something that users can manipulate, compose, inspect, and edit. All the MCT user objects are presented to users in a single unified environment (Figure 2).

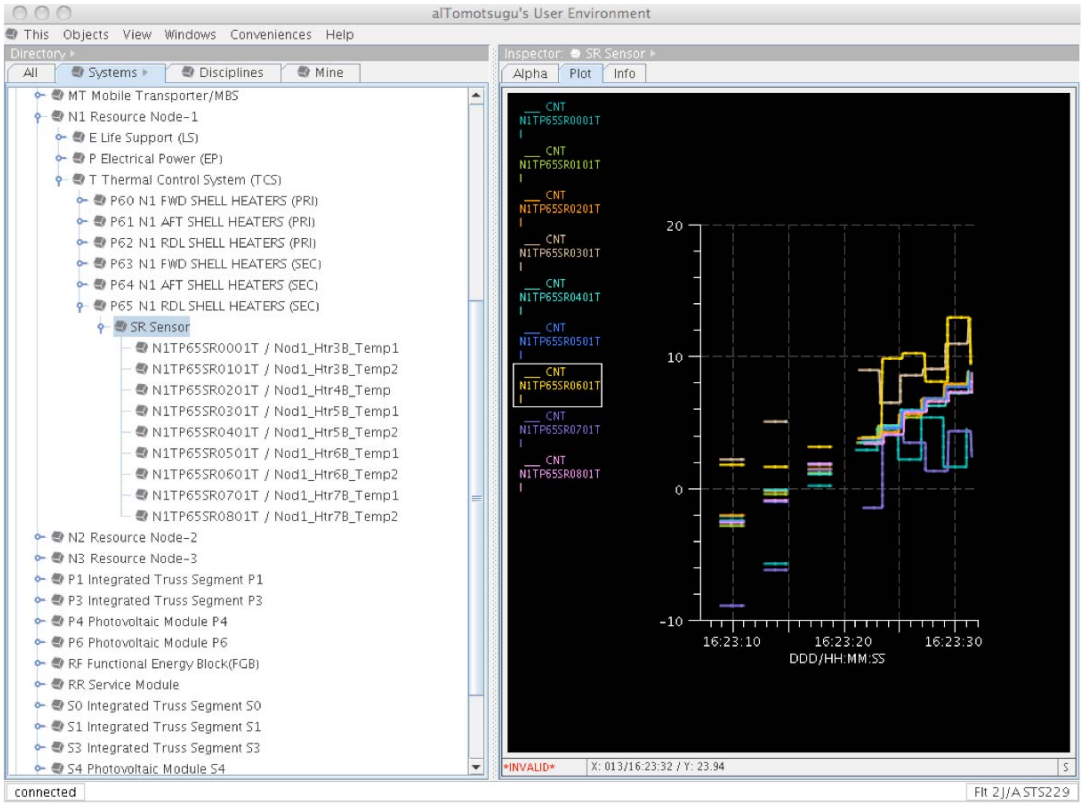
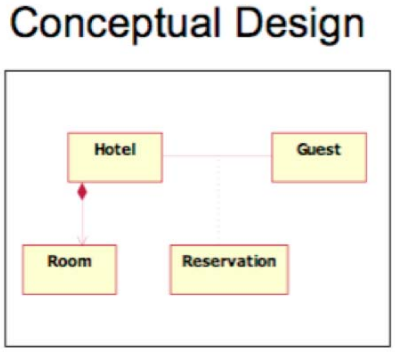



**Figure 2. MCT user environment contains all user objects**

## B. For Developers




For developers, MCT is component software built on industry standards. Developers write MCT components in Java, the user interface in Swing, and build the components as OSGi plug-ins using the MCT component model. Typically a component corresponds to a user object, thought this is not required.

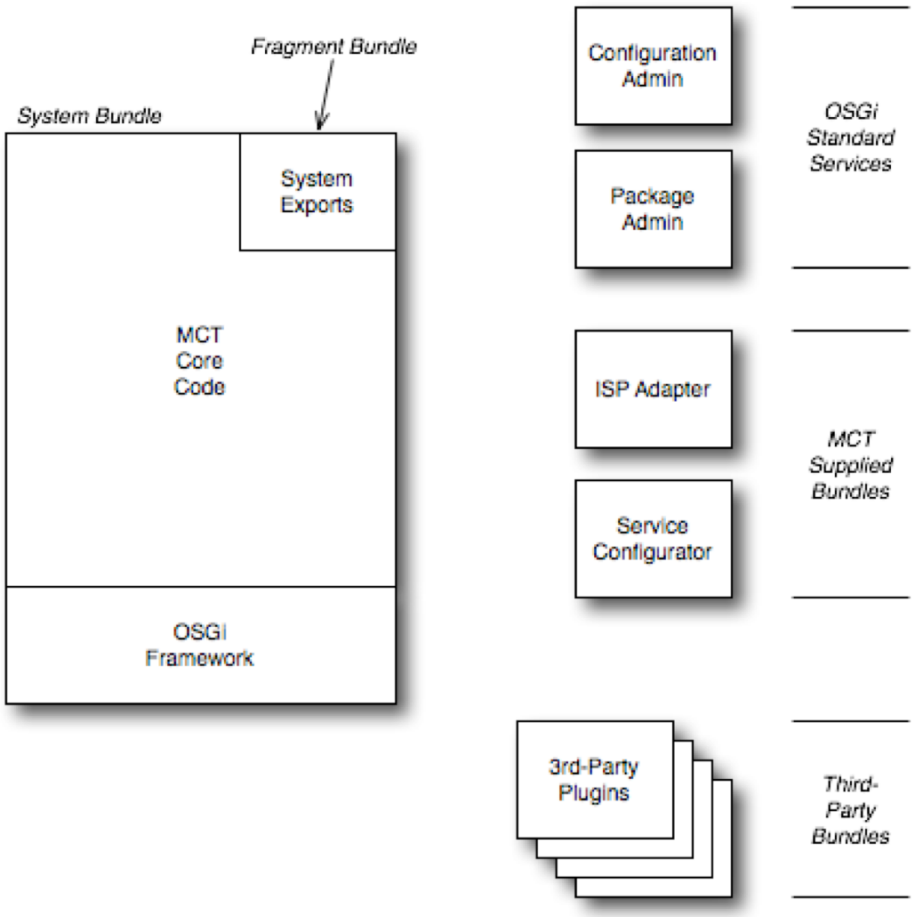
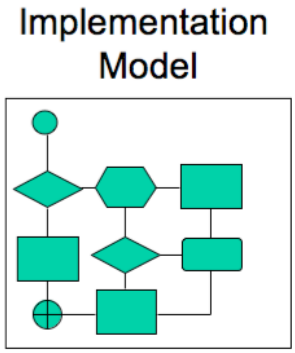



**Figure 3. MCT Architecture Overview**

### C. For Platform Service Providers




For platform service providers (PSS), MCT enables the deployment of certified software that users may reconfigure without PSS support. This potentially solves the ongoing problem of user requests for software changes that cannot be met in a timely fashion using a traditional monolithic approach to software. PSS can provide certified components, and users can compose those components without change requests or PSS support.

## II. Background and Perspective: Information Composite Approaches, Now and Then

There have been, and remain, multiple approaches to compositing information. Here we review a few of the key concepts and related projects, past and present.

## D. Historical

Here we discuss two of the many past efforts to which MCT has conceptual heritage. A comprehensive look at all the related efforts would be outside the scope of this paper.

*1. Opendoc*

Opendoc was led by Apple, with collaboration from IBM. The idea was to create a document-centric system using non-typed (meaning not tied to a specific application) documents as containers for parts tied to editors. Figure 4 contrasts conventional monolithic applications with the Opendoc approach. The Opendoc document contains a part for each type of content, such as text or an image. Each part has its own editor. Content can be edited in place in the document. The composition container is the document. The unit of composition, as perceived by the user, is the part.

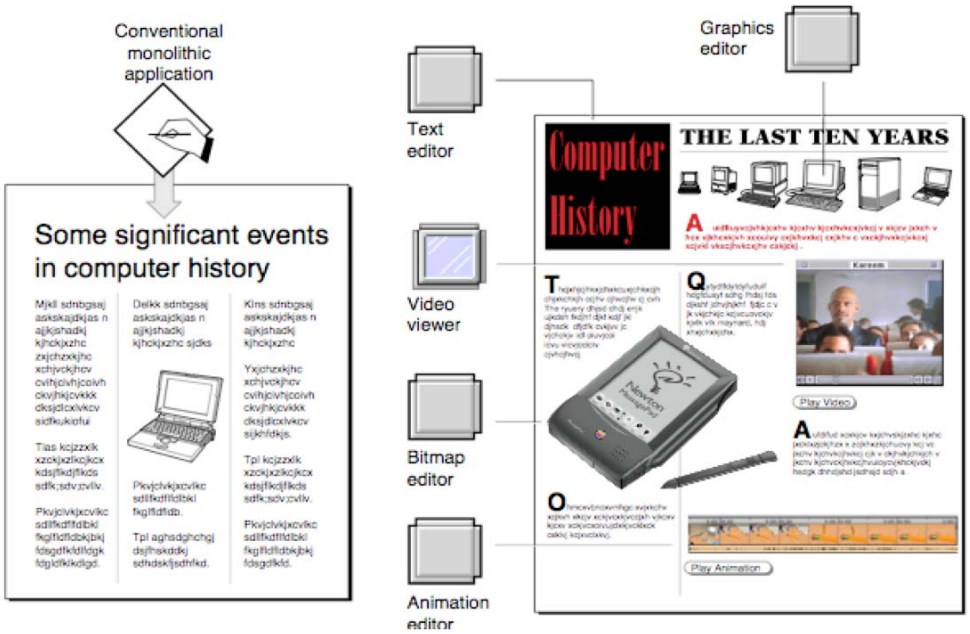



**Figure 4. Opendoc approach contrasted with monolithic applications**

MCT retains the idea of "live" objects. The practical impact of live objects is that user objects can be edited in place, without opening separate applications or editors. For example, a procedure step may be viewed, and also edited, without opening a separate procedure editor. The procedure may also be locked to prevent editing.

For more information on Opendoc see:
http://en.wikipedia.org/wiki/Opendoc

*2. IBM Common User Access*

IBM's common user access (CUA) provided guidelines for developing consistent object oriented (OO) graphical user interface (GUI) styles. The key concept as it relates to MCT is that of the OO GUI style. In an OO style GUI, the user interacts directly with "user objects" that represent entities in the applicable domain. For IBM CUA a typical example would be office objects.

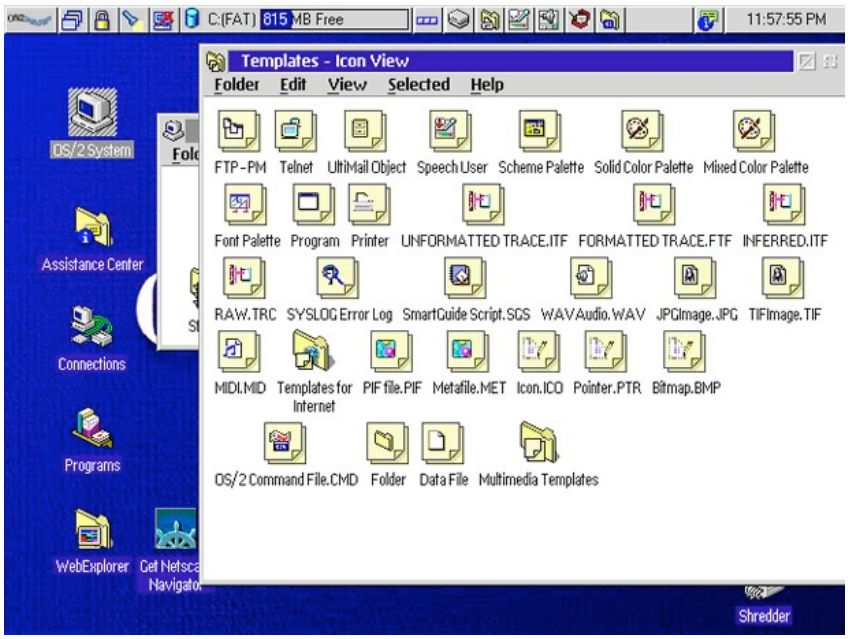



**Figure 5. IBM CUA Interface as implemented in OS-2. Note the use of template objects.**

Key aspects of an OO style GUI are:

- Users interact directly with representations of domain objects.
- User objects behave consistently as per users' domain expectations. This allows users to classify or predict object behavior, i.e., the interface does what a user familiar with the domain would expect.

Figure 5 shows a CUA interface as it was implemented in OS-2. OS-2 had limited user composability. The container was any object that acted as a container for other objects, such as a folder, or the workspace, which was the master object container.

For more information on OO GUI Styles see:
http://en.wikipedia.org/wiki/Object-oriented_user_interface

### E. Current

Coming forward into the present world, the most common example of information composition that all of us have easy access to is that of *web widgets*. Web widgets allow users to create information composites from heterogeneous data sources. Figure 6 shows an example, assembled from Google Widgets. Note the range of information types, the fixed layout grid, and the heterogeneous user interface, i.e., user interactions for each widget may be different. Web widgets are part of an open world. There is no attempt at system-wide consistency as with the objects of IBM's CUA. This allows for a wide range of creative options, but may not be optimal for consistency of user interactions or training such as we have in the domain of space operations.

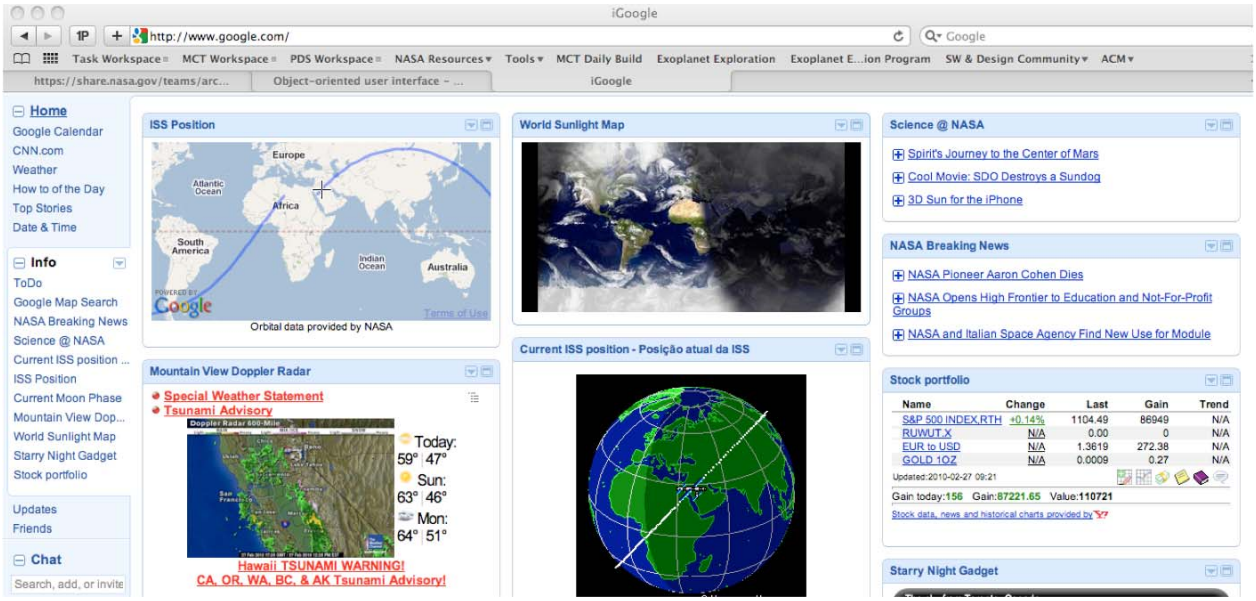
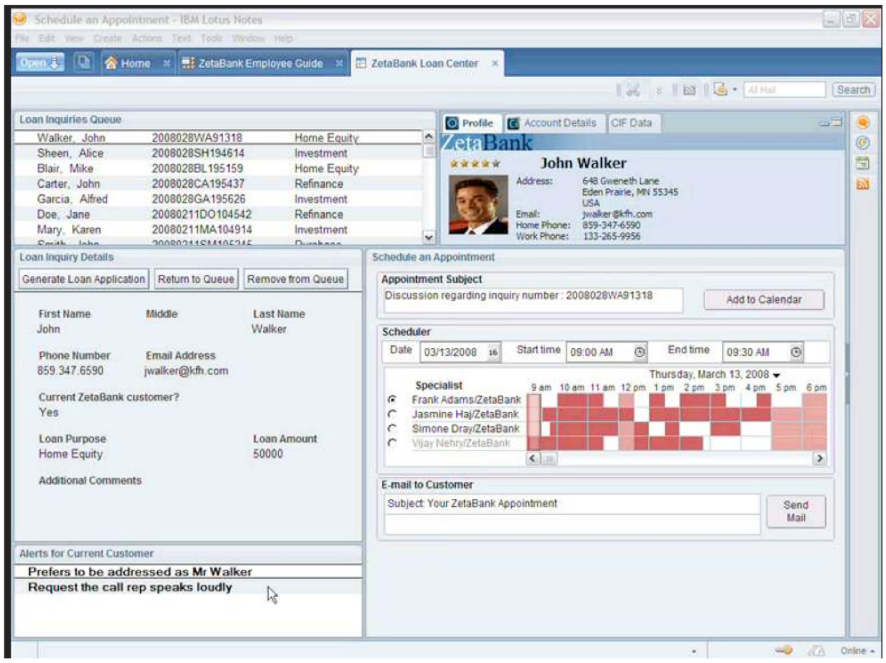



**Figure 6. A Google home page showing web widgets**

The container for web widgets is the web page. The user perceived unit of composition is the widget.

The *Eclipse Rich Client Platform* allows for the creation of composite applications by the addition of components into a consistent environment. Applications are presented to users as perspectives and views. Users can choose perspectives and views as per the choices presented to them by designers/developers.



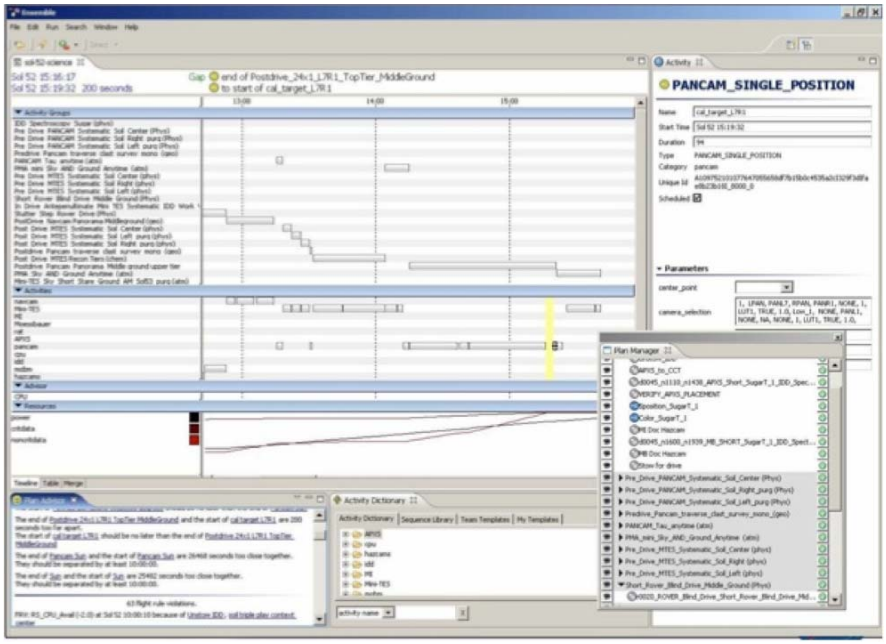



**Figure 7. Two composite applications built in Eclipse, the top showing a business application, the bottom showing JPL/ARC's Ensemble.**

For more information on Eclipse, see www.eclipse.org

## III. MCT Object Oriented Style Graphical Interface

MCT implements an object oriented graphical user interface style. The user interface is composed of relevant objects from the space operations domain. The initial versions of MCT have telemetry objects. Already in work are objects for procedures and core trajectory. All the objects are composable, with consistent behaviors. The user unit of composition is the user object. The MCT container is called a collection.

Now, let's take a look at the MCT user environment, the contained objects, and some compositions.

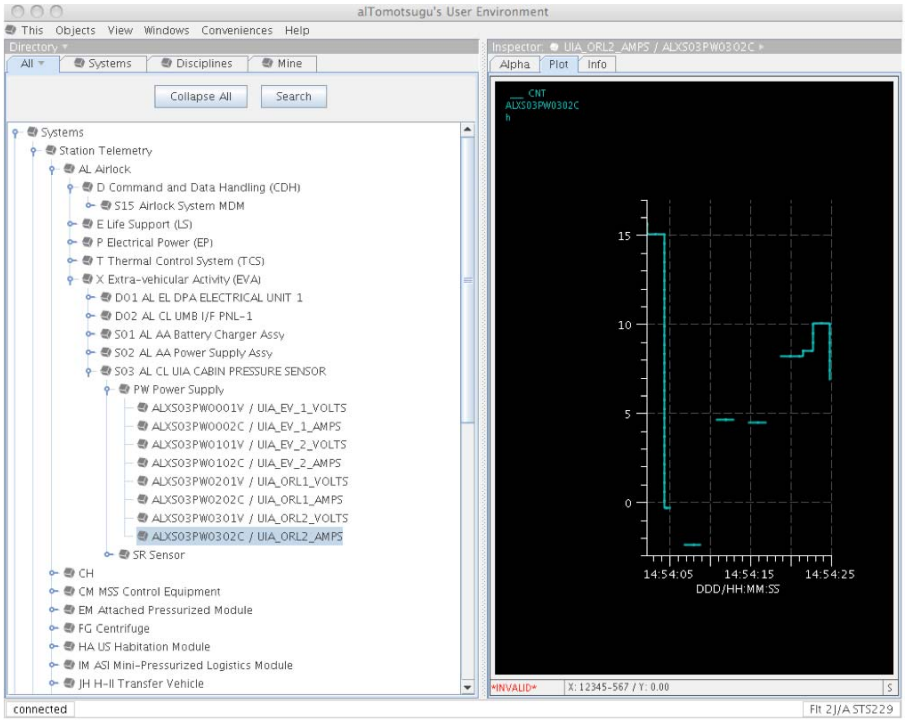



**Figure 8. The MCT User Environment Contains all user objects.**

### F. The User Environment Contains all user objects

The user environment in MCT is the "root" object, meaning that it contains all available user objects. Note that this does not mean root in the object oriented sense of inheritance; rather, it refers to containment. The user environment is the master container of all user objects. See Figure 8. For the user, this means that, if they need to find any user object, they have one place that they know will always have it.

The inspector on the right shows details of any object. Note that the inspector can display multiple views. This implements a key principle of MCT—that the same object may be viewed in different ways. For telemetry, current view options are alphanumeric, plot and info.

Objects may be located by browsing or searching the user environment. Given that there are tens of thousands of user objects, we expect search to be used frequently. See Figure 9.

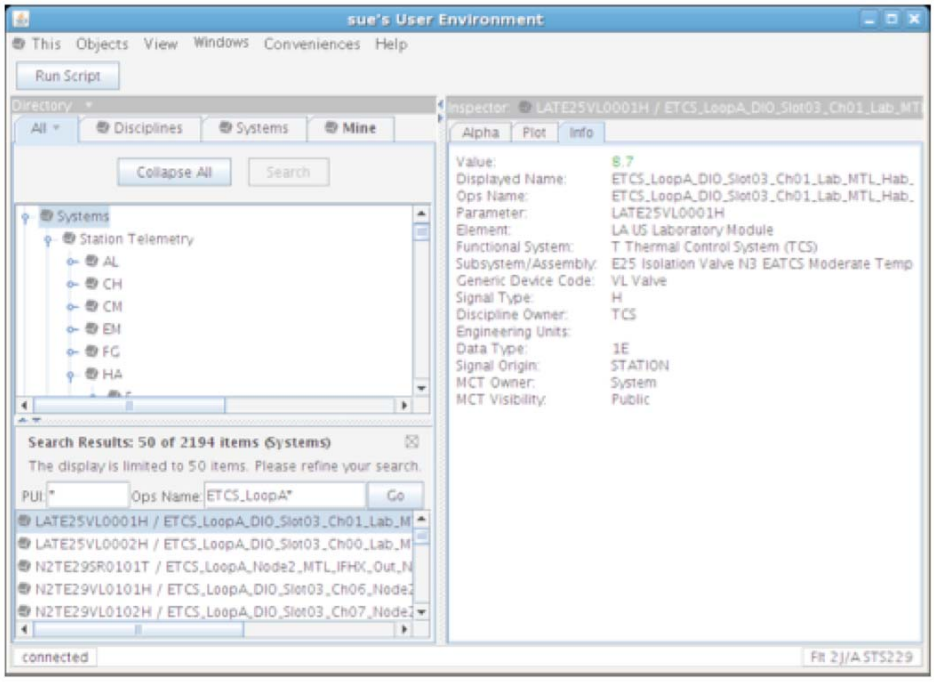



**Figure 9. Searching the MCT user environment**

### G. The Same Thing May be Viewed Differently

A user object is always the same "thing," the underlying properties remain the same, but it may be viewed differently. Figure 10 shows an example of this for telemetry objects.

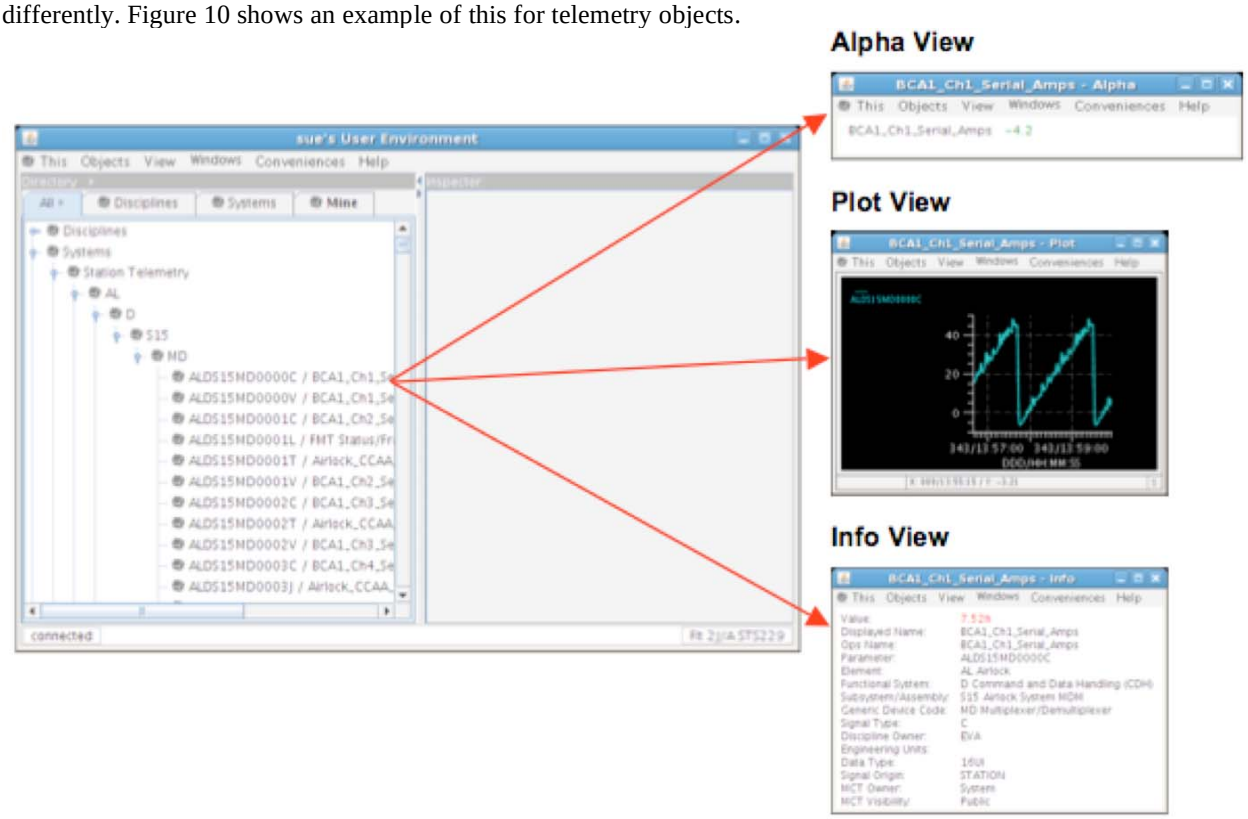



**Figure 10. A telemetry user object shown in alpha, plot, and info views**

## H. Displays are Compositions of Objects, Grouped into Collections

Figure 11 shows an a simple user composition. On the left is a directory showing every user object in the composition. In the center is the canvas area, that's where the user objects are composed. On the right is the inspector. Displays may be built up through multiple levels of compositions.

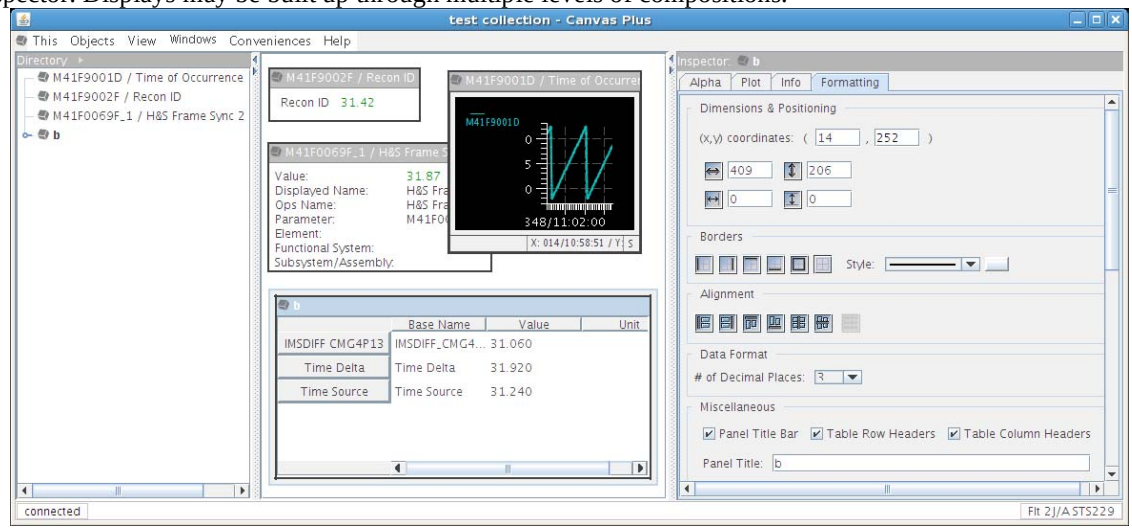



**Figure 11. A simple user-assembled composition**

## I. A User Object is the Same Thing Everywhere

A fundamental principle of MCT is that a user object is the same thing everywhere. In practice this means that user objects may be composed into any kind of display, anywhere, and the underlying object properties do not change. A piece of telemetry may be shown as an alphanumeric on a telemetry display, or composed into a procedure on another. It's still the same thing. Figure 12 shows one example of a single telemetry point in different collections, with different views.

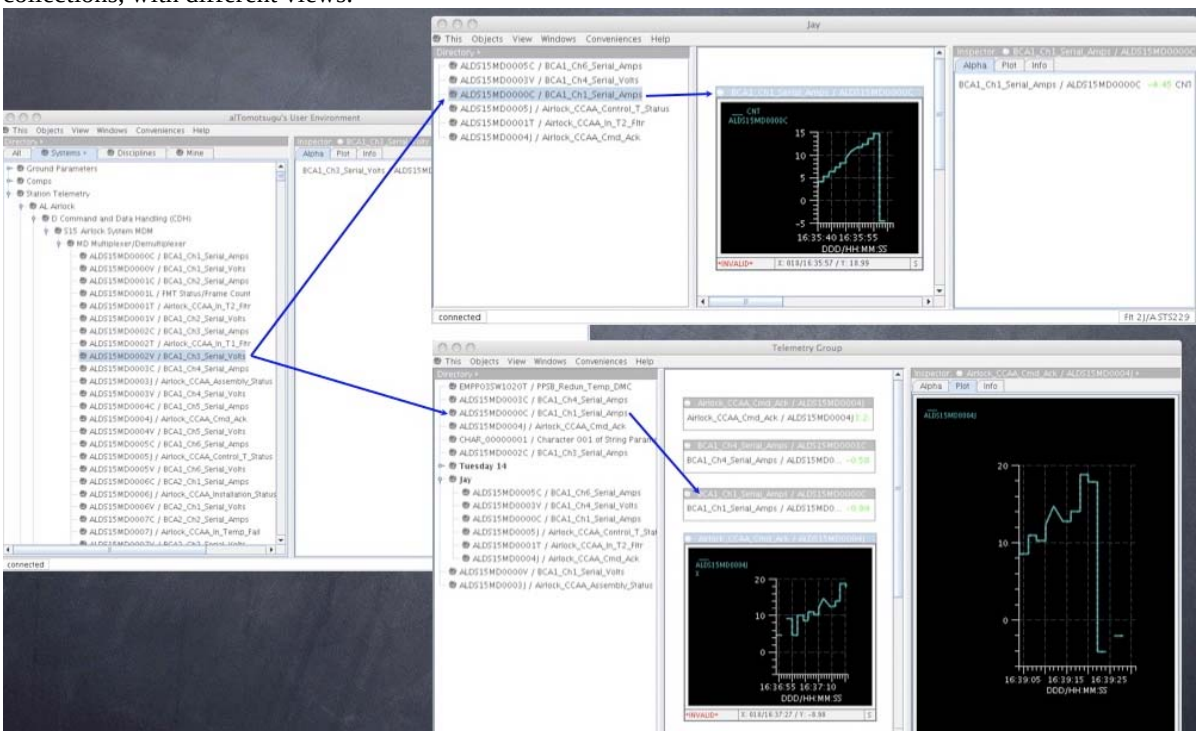


**Figure 12. A user object is the same thing everywhere**

## IV. Capability Phasing

The initial MCT deliveries contain user objects for telemetry and monitoring. A developers toolkit enables third parties to develop user objects (which developers would refer to as components). Third-party code currently under development includes procedures, core trajectory, and prototype command objects.

## V. MCT Project Status

MCT completed the initial prototypes and testing in JSC's Operations Technology Facility (OTF) in FY07. MCT was funded as a formal project beginning in FY08. The initial Module One delivery took place in Summer of 2009. Iterations are delivered to JSC roughly every five weeks. MCT has "graduated" from the OTF to the first phase of installation in the MCC, in the Common Development Environment (CDE). After testing and verification are complete in CDE, the next step is to move onto the MCC opsnet, into uncertified operations. This will provide MCC-wide access to flight controllers so that they can build displays and use MCT for flight following. This is expected to take place in the Summer of 2010. Following several months of operations in the uncertified environment, MCT should graduate to operations in FY11.

## VI. Acknowledgements

**J. The MCT Team at ARC**

Sarah Hobart, Nija Shi, Tom Dayton, Mark Rose, Alan Tomotsugu, Antonio Si, Irene Smith, Alex Voskoboynic, Chris Webster, Peter Jarvis, Laura Baalman, Madelyn Quinol, Igor Lyandres, Sue Blumenberg

**K. The MCT users and team at JSC**

Kevin Jennings, Kevin Taylor, Mark Hussey, Ed Turcott

**L. The MCT Interns**

Ketaki Borkar, Benson Hong