Mission Control Technologies: A New Way of Designing and Evolving Mission Systems

Jay Trimble [1]and Joan Walton [2]
NASA Ames Research Center, Moffett Field, CA 94035

and

Harry Saddler [3]
QSS/NASA Ames Research Center, Moffett Field, CA 94035

Jay: instead of tracking changes, I added a comment like this wherever I made a significant change.

Some of the illustrations are too low-res to print well. I changed a few of them, but I don't have all of the files. If you send me the Squeak screenshots and figure 2, I can fix 'em up (remember when you put figures in a paper, they're going to be printed at 300-600 (or even 1200) dpi, so if they're pasted in at 72 dpi, they won't be very legible).

Also: I think it works better to have embedded images without frames or wrapping... otherwise, they're always floating around in the text. I took 'em out.

If you want me to improve t he formatting, let me know.

I. Introduction

Current mission operations systems are built as a collection of monolithic software applications. Each application serves the needs of a specific user base associated with a discipline or functional role. Built to accomplish specific tasks, each application embodies specialized functional knowledge and has its own data storage, data models, programmatic interfaces, user interfaces, and customized business logic. In effect, each application creates its own walled-off environment. While individual applications are sometimes reused across multiple missions, it is expensive and time consuming to maintain these systems, and both costly and risky to upgrade them in the light of new requirements or modify them for new purposes. It is even more expensive to achieve new integrated activities across a set of monolithic applications.

These problems impact the lifecycle cost (especially design, development, testing, training, maintenance, and integration) of each new mission operations system. They also inhibit system innovation and evolution. This in turn hinders NASA's ability to adopt new operations paradigms, including increasingly automated space systems, such as autonomous rovers, autonomous onboard crew systems, and integrated control of human and robotic missions.

Hence, in order to achieve NASA's vision affordably and reliably, we need to consider and mature new ways to build mission control systems that overcome the problems inherent in systems of monolithic applications. The keys to the solution are *modularity* and *interoperability*. Modularity will increase extensibility (evolution), reusability, and maintainability. Interoperability will enable composition of larger systems out of smaller parts, and enable the construction of new integrated activities that tie together, at a deep level, the capabilities of many of the components. Modularity and interoperability together contribute to flexibility.

The Mission Control Technologies (MCT) Project, a collaboration of multiple NASA Centers, led by NASA Ames Research Center, is building a framework to enable software to be assembled from flexible collections of components and services.

II. Background

The need for MCT became apparent during software development, user observation, and user requests from the Mars Exploration Rover (MER) mission operations technology infusion projects, in which Ames and the Jet

[1] Computer Scientist, Intelligent Systems Division, NASA Ames Research Center Moffett Field CA USA 94035
[2] Insert Job Title, Department Name, Address/Mail Stop, and AIAA Member Grade for second author.
[3] Insert Job Title, Department Name, Address/Mail Stop, and AIAA Member Grade for third author.

Propulsion Laboratory (JPL) collaborated to deploy software into MER operations. The software for mission operations is, of necessity, designed based on requirements that are specified several years before launch. By the start of training, the software gets its first serious shakedown in a realistic operations setting.

During training, we learn better ways to operate. Processes are improved, new features are added, and existing features are changed. However, it is too late to make any significant changes to software by this point, leaving labor-intensive operational workarounds, add-on applications, scripts, and other "glueware" patches to software as the only options for improvement. In several cases during training and operations for MER, users requested integration of new functionality, but the limitations of current software technologies prevented us from being able to meet these requests.

A. Current Practice: Predetermined Functional Groupings

Current software development practice creates predetermined functional groupings of features which are presented to the user as applications, which operate as their own sealed-off worlds. Data interoperability is limited and is achieved through file transfers or cut-and-paste operations through clipboards. Reconfiguration is only possible through upgrades and re-release, limited user interface customizations, or the addition of plug-ins. Fundamental reconfiguration to match evolving processes and workflow is not possible.

Figure 1 illustrates these isolated worlds.

Updated this illo with better-quality one:

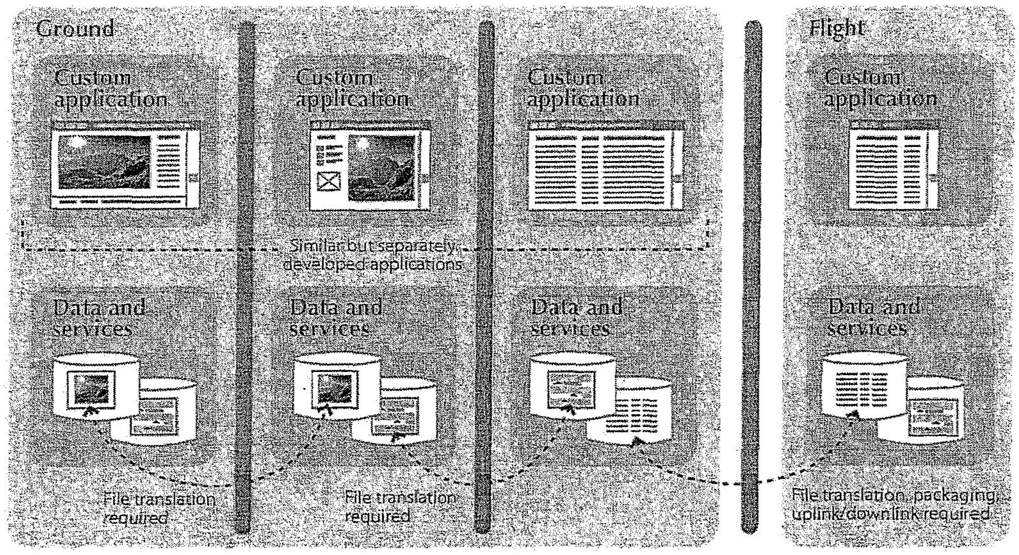


Figure 1. Traditional application development with monolithic applications. Communication and interoperability take place across "walls," in this case in the form of file transfers.

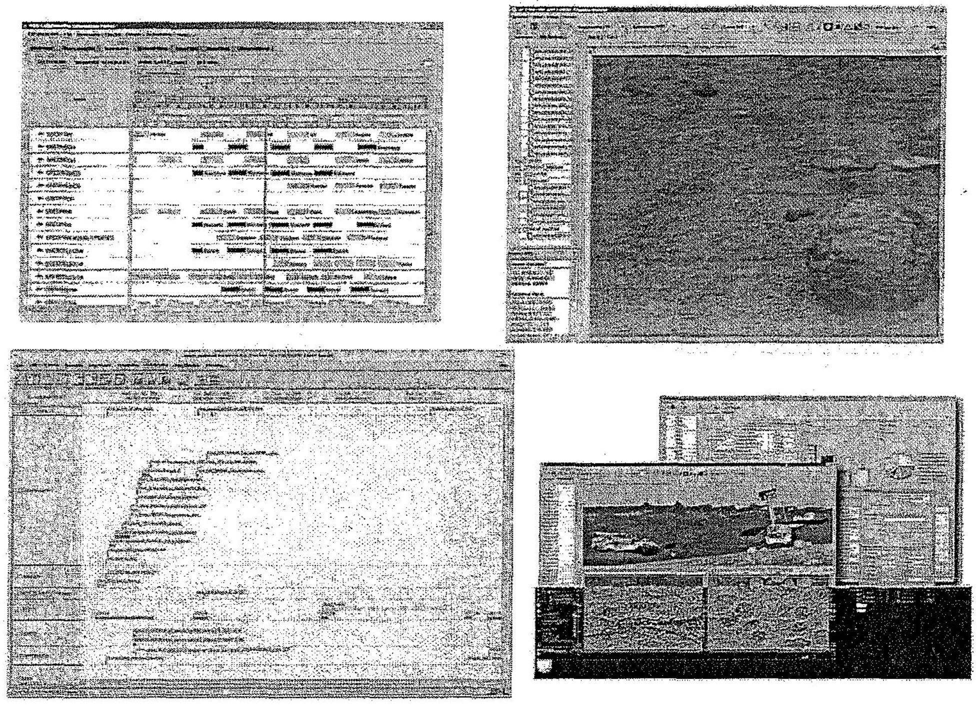


Figure 2. *Duplication of functionality and inconsistent* user interfaces in MER Ground System tools, developed across centers

Since monolithic, isolated applications cannot easily be used together, it becomes necessary to add similar features to each; over time these applications thus tend to become bloated with inconsistently duplicated features, adding to the difficulty of using and maintaining them. Another by-product of this process is inconsistent user interfaces, which creates added risk of operational errors and increases training time for operations personnel.

B. Related Industry History

There have been attempts to develop software that is not composed from large, walled-off monolithic chunks. Dating back to original and well-known research at Xerox PARC that led to the modern computing era, the Smalltalk language organized functionality into objects; new features and capabilities were added dynamically simply by adding new objects. There were no applications as we know them today. The Xerox Star, the first commercially available computer with a graphical user interface, had a purely object-oriented desktop and compound documents. These systems were created at a time of limited hardware performance, which limited software design options.

Subsequently, Apple Computer and IBM collaborated on a project called OpenDoc. OpenDoc was an attempt to replace applications with a document-centric approach wherein functional, content-bearing objects, or "parts," could be assembled into fully functional "documents." Figure 3 shows the OpenDoc concepts of building documents from "parts" and "editors," contrasted with monolithic applications.

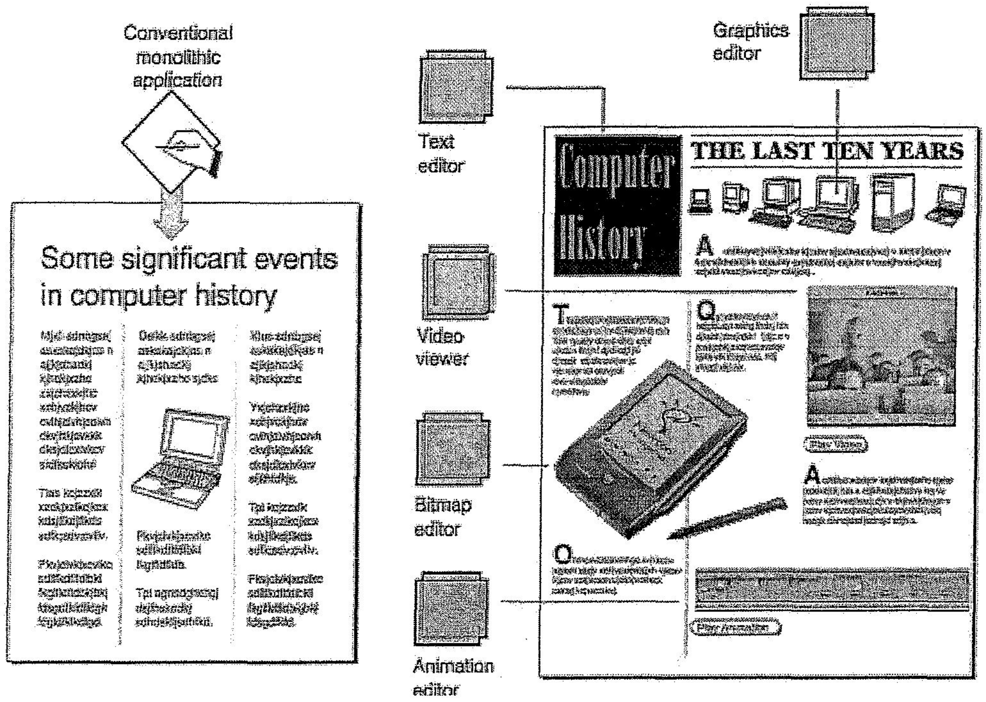


Figure 3. OpenDoc and Monolithic Applications Compared

It was these projects from industry, combined with our experience on the MER Project, that inspired MCT. We started with a simple idea: instead of building software from monolithic applications, build *components* that can be easily assembled and recombined as needed to meet user, task, and mission requirements. Such components could be *composed* to create complex, task-specific *compositions*, replacing traditional applications. By making the components composable, we can achieve interoperability, including the capability to combine components developed across organizations. Modularity and flexibility are achieved by making components fine-grained. This allows for flexible reconfiguration of composable components, even after the start of training and operations. Figure 4 shows the MCT concept of assembling applications and operational environments from components on top of a supporting framework.

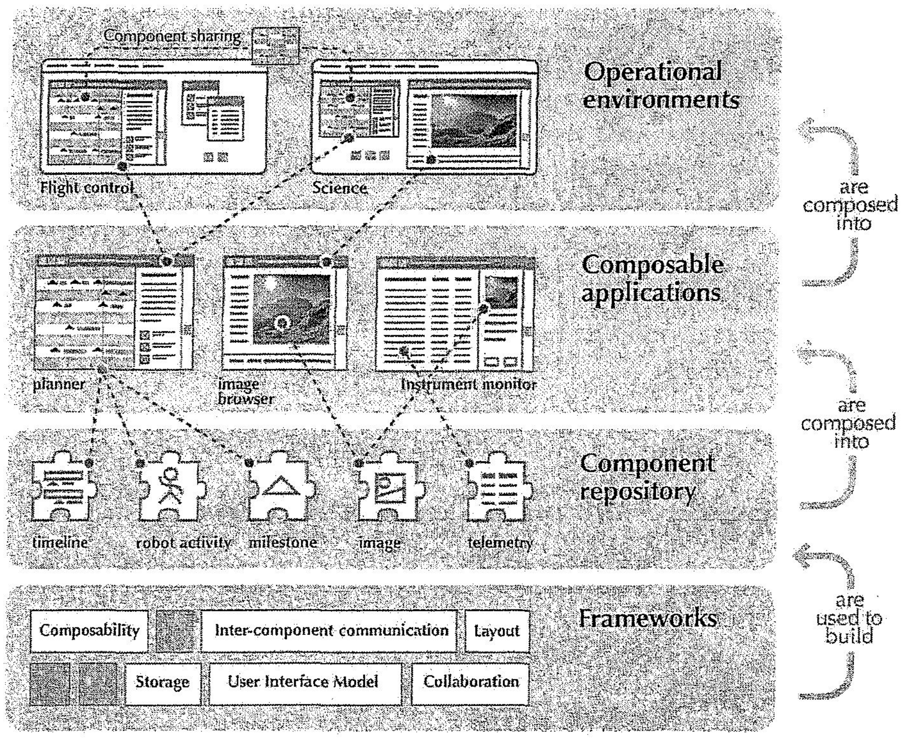


Figure 4. MCT applications and environments are assembled from components.

III. MCT Architecture and Frameworks

MCT consists of a system architecture that supports the deployment and use of distributed fine-grained components; a user interface architecture that helps users create, manage, and understand mission data, and to organize their tools to suit their tasks and preferences; a set of frameworks enabling the development and operation of components; and information models that support component interoperability. This infrastructure is implemented in part as a set of plug-ins atop the Eclipse open source development platform (see www.eclipse.org).

A. An Architecture for Composable User Interfaces

Background

A well-known and time-tested user interface architecture is the model-view-controller (MVC) design pattern. MVC, invented at Xerox PARC in the early 80's, has evolved over time, and is still in use today. The fundamental idea is to create flexibility in software development by separating the presentation of the interface to the user from the underlying model, i.e. separate the user interface front end (view) from the application backend (model) and the input which is handled by the controller. As a simple example consider data accountability as shown in figure X.

<Figure X - insert diagram showing accountability model, with multiple views and a controller>

What did you have in mind for figure X?

In "traditional" object oriented architectures, the fundamental building blocks are objects. In the MCT architecture, the fundamental building block is a *component*. MCT components are objects with a number of additional characteristics supporting flexible reuse, the ability to compose user interfaces during run-time, and the use of information models to control system behavior. In MCT, a component may be presented to the user in different ways through *representations* which serve as the components' user interfaces. One component may have multiple representations; this both increases component reuse and allows the same component to be used in multiple locations, which improves consistency and enables remote collaboration. So far we have the capability to build a component and show it in different ways (representations) and locations (distributed components).

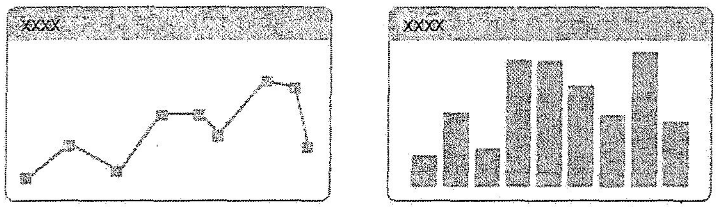
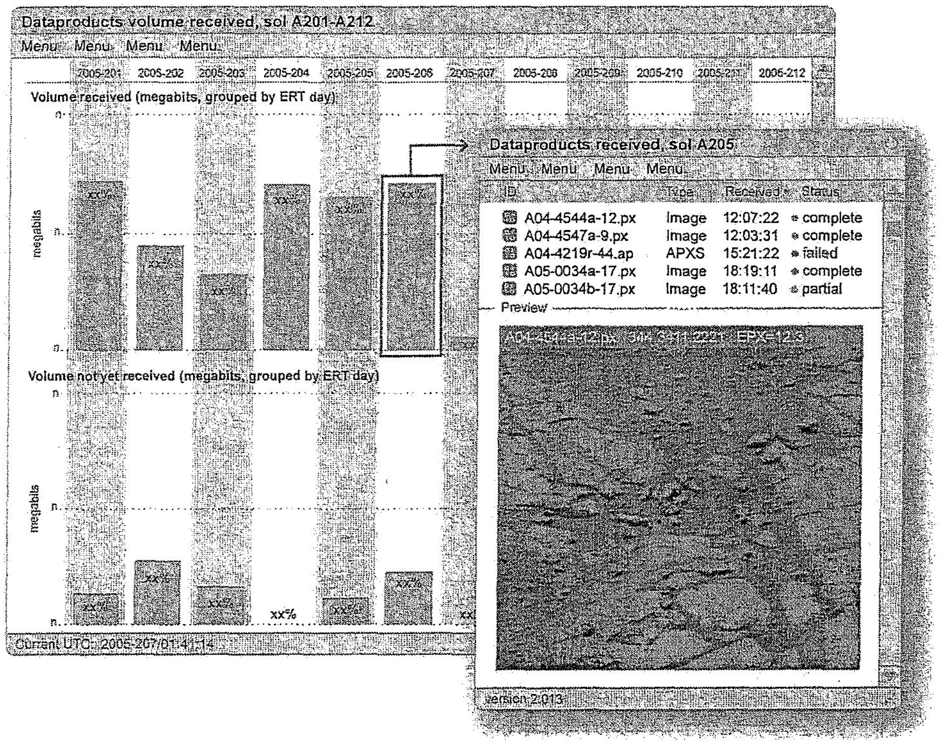


Figure 5. Multiple representations of the same data

Updated illo above; also moved it from where it was — after the accountability illo.

Revised following 2 paragraphs:

In the MCT architecture, components have *roles* which define the attributes and behaviors of components in accordance with the information model. MCT also uses *composition policies* to specify the kinds of compositions that can be created, and the interaction between components in a composition.



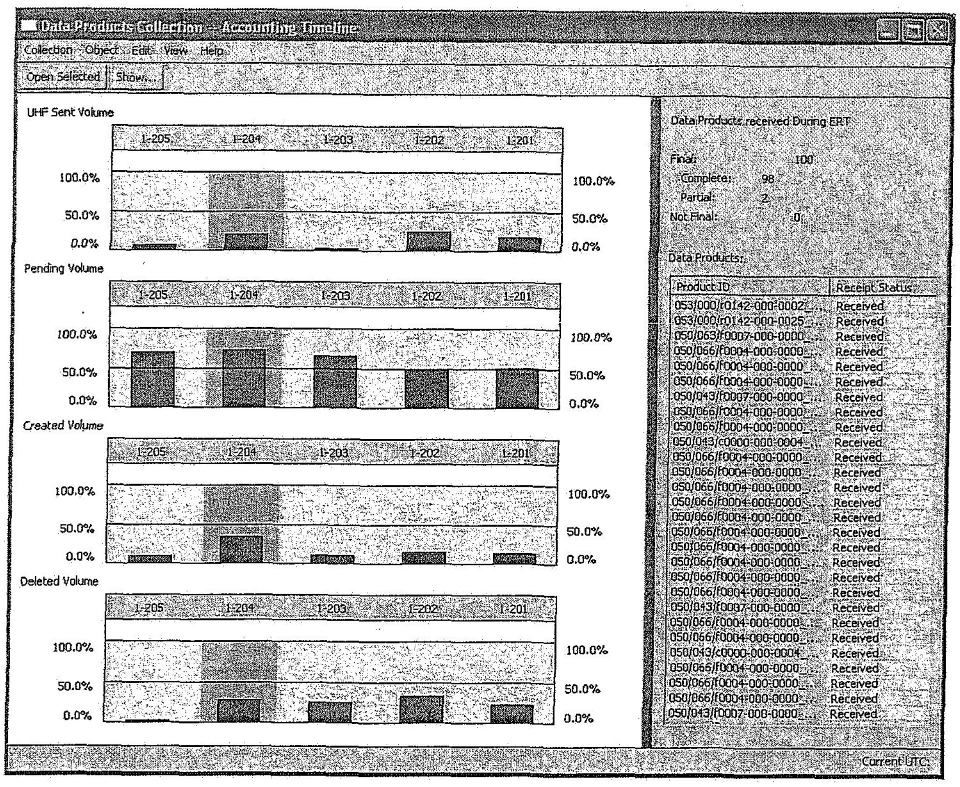
ID	Type	Received	Status
A04-4544a-12.px	Image	12:07:22	complete
A04-4547a-9.px	Image	12:03:31	complete
A04-4219r-44.ap	APXS	15:21:22	failed
A05-0034a-17.px	Image	18:19:11	complete
A05-0034b-17.px	Image	18:11:40	partial

Some changes at the end of this paragraph

Now, let's look at how that accountability model would look in the MCT architecture. Figure X shows two compositions that were assembled for different purposes; one to track data accountability, and the other to examine scientific data. The bar chart bars, list of images, and the thumbnail image are all representations of the same kind of component — a science dataproduct. The user is able to "inspect" the bar to see what dataproducts contribute to it, and then view individual data products; this sort of exploratory capability is inherent to all components because in MCT a component is always "live." Unlike current environments where documents are assembled from multiple pieces that are cut and pasted in, MCT compositions are assembled from components that may be edited in place at any time.

Figure 6. Exploring data using multiple representations

Added following paragraph, trimmed caption:



Data Products received During ERT

Final:	100
Complete:	98
Partial:	2
Not Final:	0

Data Products:

Product ID	Receipt Status
053/000/r0142-000-0002_...	Received
053/000/r0142-000-0025_...	Received
050/063/f0007-000-0000_...	Received
050/066/f0004-000-0000_...	Received
050/066/f0004-000-0000_...	Received
050/066/f0004-000-0000_...	Received
050/043/f0007-000-0000_...	Received
050/066/f0004-000-0000_...	Received
050/066/f0004-000-0000_...	Received
050/043/c0000-000-0004_...	Received
050/066/f0004-000-0000_...	Received
050/066/f0004-000-0000_...	Received
050/066/f0004-000-0000_...	Received
050/066/f0004-000-0000_...	Received
050/066/f0004-000-0000_...	Received
050/066/f0004-000-0000_...	Received
050/066/f0004-000-0000_...	Received
050/066/f0004-000-0000_...	Received
050/066/f0004-000-0000_...	Received
050/066/f0004-000-0000_...	Received
050/043/f0007-000-0000_...	Received
050/066/f0004-000-0000_...	Received
050/066/f0004-000-0000_...	Received
050/043/c0000-000-0004_...	Received
050/066/f0004-000-0000_...	Received
050/066/f0004-000-0000_...	Received
050/066/f0004-000-0000_...	Received
050/043/f0007-000-0000_...	Received

Figure 7. Data accountability representations from the 2005 pilot

Figure X and Y show data accountability representations from the 2005 pilot built in collaboration with JPL. The *pilot showed the capability to build composable systems on top of Eclipse and connect to existing services and data* sources.

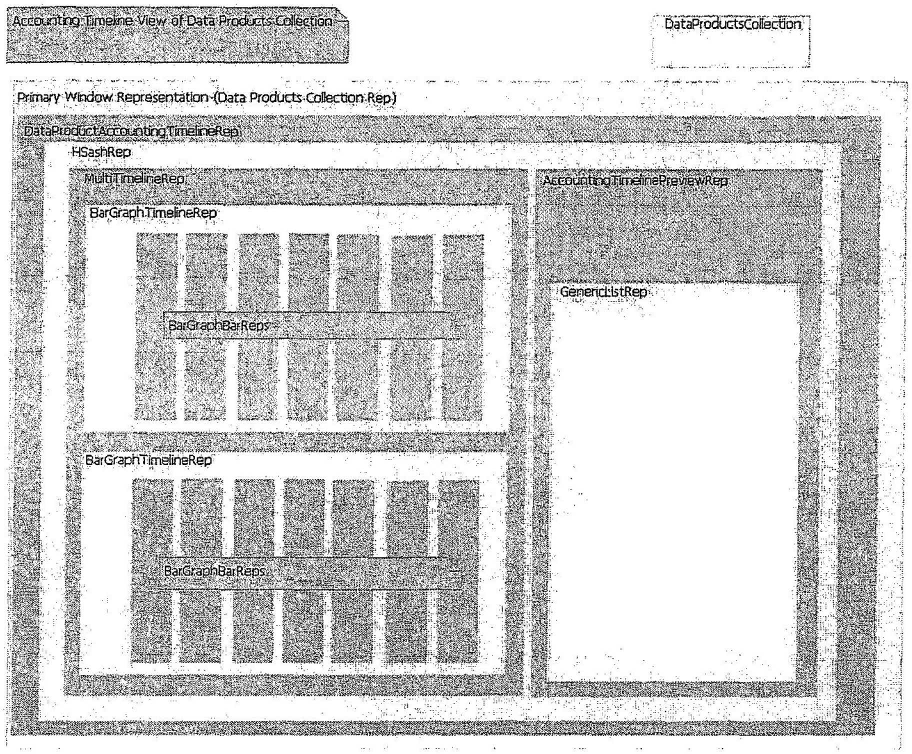
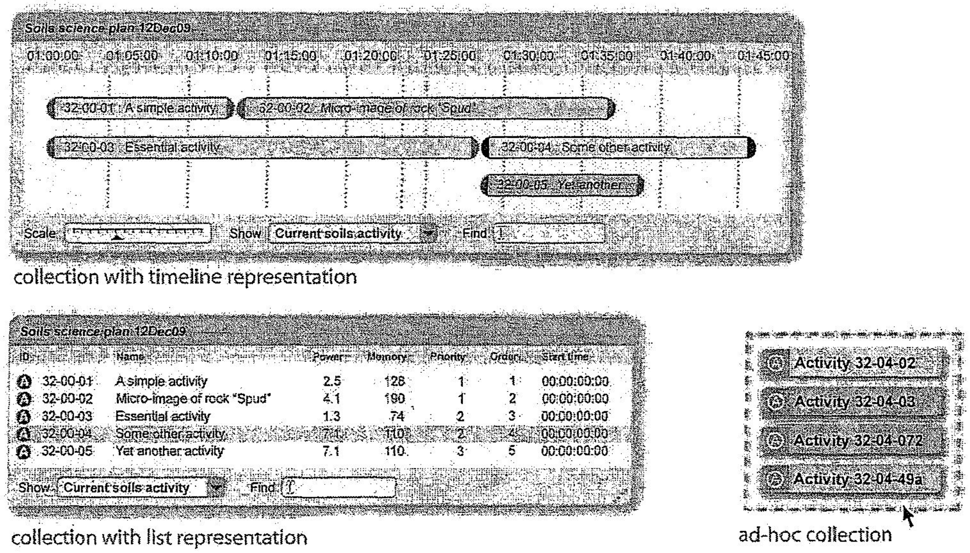

Accounting Timeline View of Data Products Collection
DataProductsCollection
Primary Window Representation (Data Products Collection Rep)
DataProductAccountingTimelineRep
HSashRep
MultiTimelineRep
AccountingTimelinePreviewRep
BarGraphTimelineRep
BarGraphBarReps
GenericListRep
BarGraphTimelineRep
BarGraphBarReps

Figure 8. Representation detail from the data accountability pilot shown in Fig. 7

Added following paragraph:

Another example of multiple represenentation in MCT is in the way it can present *collections*. A collection is simply a set of components. Collections are often displayed as lists, but depending on the type of components in the collection (and the user's task), may be presented in other ways. Figure 10 shows a collection of rover activities shown as a filterable list, perhaps being used to quickly find related activities, and as a timeline which shows and manages the temporal relationships among the activities, in this example being used to create rover command sequences.


Soils science plan 12Dec09
01:00:00
01:05:00
01:10:00
01:15:00
01:20:00
01:25:00
01:30:00
01:35:00
01:40:00
01:45:00
32-00-01 A simple activity
32-00-02 Micro-image of rock "Spud"
32-00-03 Essential activity
32-00-04 Some other activity
32-00-05 Yet another
Scale
Show
Current soils activity
Find

collection with timeline representation

Soils science plan 12Dec09

ID	Name	Power	Memory	Priority	Group	Start time
Ⓐ 32-00-01	A simple activity	2.5	128	1	1	00:00:00:00
Ⓐ 32-00-02	Micro-image of rock "Spud"	4.1	190	1	2	00:00:00:00
Ⓐ 32-00-03	Essential activity	1.3	74	2	3	00:00:00:00
Ⓐ 32-00-04	Some other activity	7.1	110	2	4	00:00:00:00
Ⓐ 32-00-05	Yet another activity	7.1	110	3	5	00:00:00:00

Show: Current soils activity Find:

collection with list representation

ad-hoc collection

Figure 9. Examples of representations and collections

B. Information Models

MCT uses information models, in the form of ontologies, to support component *interoperability*. To quote Pollock and Hodgson [insert ref.], "Interoperability ... means using loosely coupled approaches to share or broker software resources while preserving the integrity and native state of each entity and each data set." MCT components are designed to be separately engineered, validated, and deployed, but capable of intelligently interacting. At the scale envisioned for MCT, component *integration*—where components are assimilated into a larger whole through tight bindings such as APIs—would be unmanageable and undesirable.. Instead, connections between components are captured in ontologies, which can express any number of relationships. In MCT, information models capture domain knowledge about the "real-world" information that is being displayed or manipulated by the component (data types, categories, translators, units, groupings, etc.) Information models are also employed to describe system aspects such as services, structure and content of messages, and composition policies. With this information represented explicitly, rather than implicitly in the form of method names and argument lists, the MCT frameworks can incorporate various inferencing techniques to guide the behavior of the components based on the information models.

IV. Prototypes and First Pilot

In 2005, the MCT Project developed both initial prototypes and a pilot application. The purpose of the prototype was to test the component model and user interface architecture across a representative set of applicable domains. We chose telemetry, procedures, and planning. The purpose of the pilot was to build the first working version of what will evolve into the deployed mission framework, and to build an application and connect it to mission data.

A. The Preliminary Conceptual Prototype

A preliminary prototype for exploring fundamental MCT concepts was built using Squeak, a variant of Smalltalk (see www.squeak.org). Whereas the pilot effort focused on a narrow set of functionality in depth, this prototype focused on a broad set of functionality but did not go into depth in any particular area. We started with the fundamental idea that the user should not have to start an application or be restricted by any set of predetermined inflexible functionality. Rather, the user would be presented with a component repository, or a user environment, from which compositions may be put together.

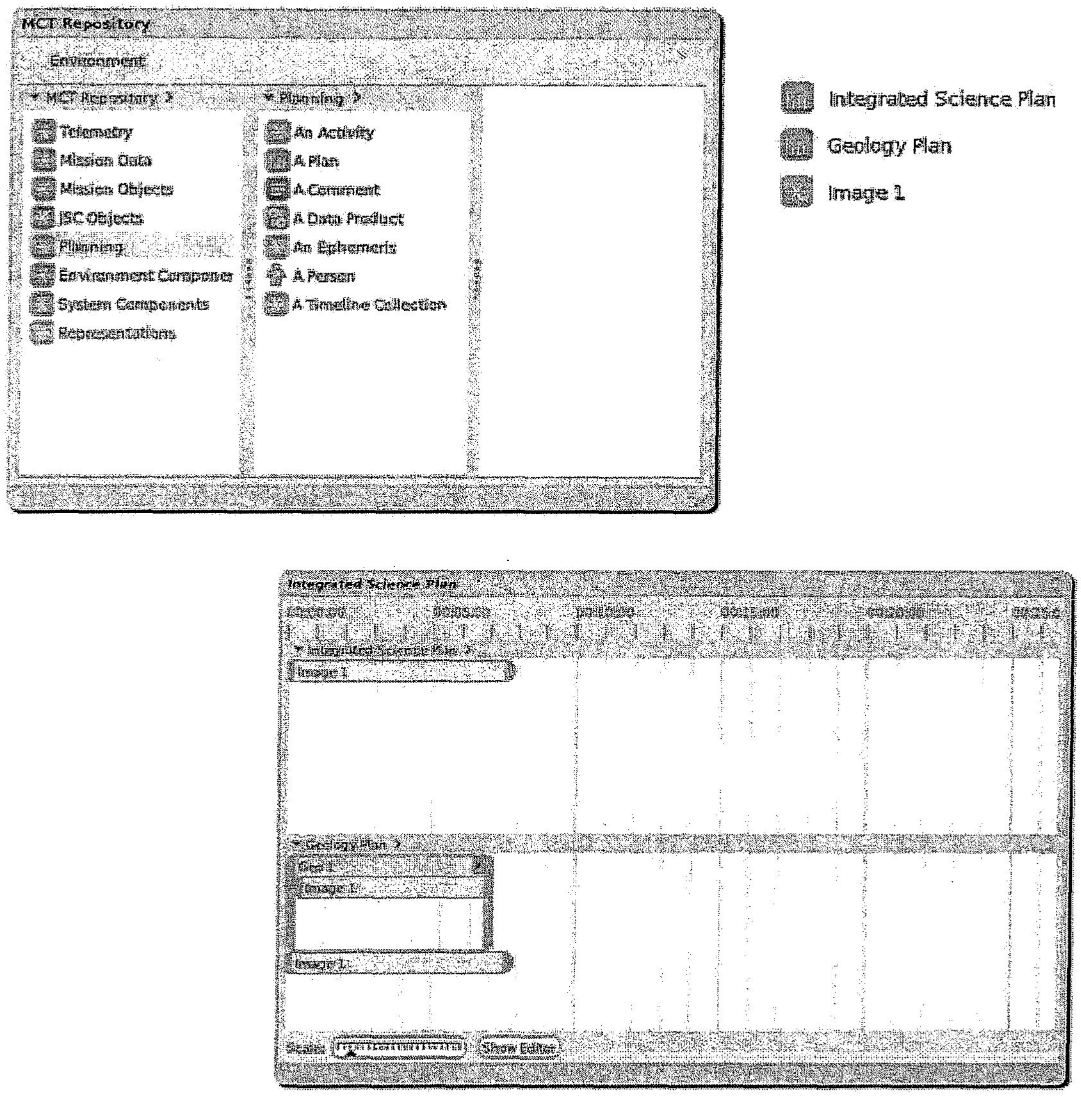


Figure 10. A simple composed plan

Replace fig. 10 with a better figure

The pilot validated the capability of the MCT component model to build compositions crossing different domains, including planning, telemetry and procedures. We built components and representations for timelines, activities, telemetry data points and procedural elements. We were able to combine them to meet not only current system requirements for such systems, but also in ways not possible with current systems.

Figure 10 shows the prototype component repository and a simple plan composition. The plan is composed of a timeline - a simple component that meets a near universal space mission requirement to show data in a time-based graphical representation, activities and a plan within a plan. The composition was created by dragging and dropping components from the repository. This composition is much like a traditional timeline. However, many more combinations are possible. The plan may have as many or few timeline components as needed. Any piece of data with a time representation may be embedded in any part of the plan. Any component in the plan may be displayed as part of another composition, such as a users notebook. Non-time based respresentations, such as sticky notes, may also be embedded in a plan, depending on composition policies as implemented by the mission. A representation may appear in multiple locations in the plan, or in other collections and compositions. Extension to multiple locations is possible.

B. A Composition for Telemetry and Procedures

Expanding on our initial explorations in timelines and planning compositions, we moved into the domain of telemetry and procedures. Current International Space Station (ISS) and Shuttle operations utilize operational setups with telemetry organized into functional collections built around system models. Verification and validation of procedures requires correlation of expected telemetry values based on executed procedural steps. The procedures and the telemetry screens have been separate worlds. As MCT is about breaking down these types of artificial boundaries, we further validated the MCT component model and UI architecture by putting together compositions such as Fig. 11. The compositions shown demonstrate the capability to build "traditional" telemetry and monitoring screens, as well as the capability to combine telemetry and procedures into the same composition. These compositions were built using ISS displays and procedures.

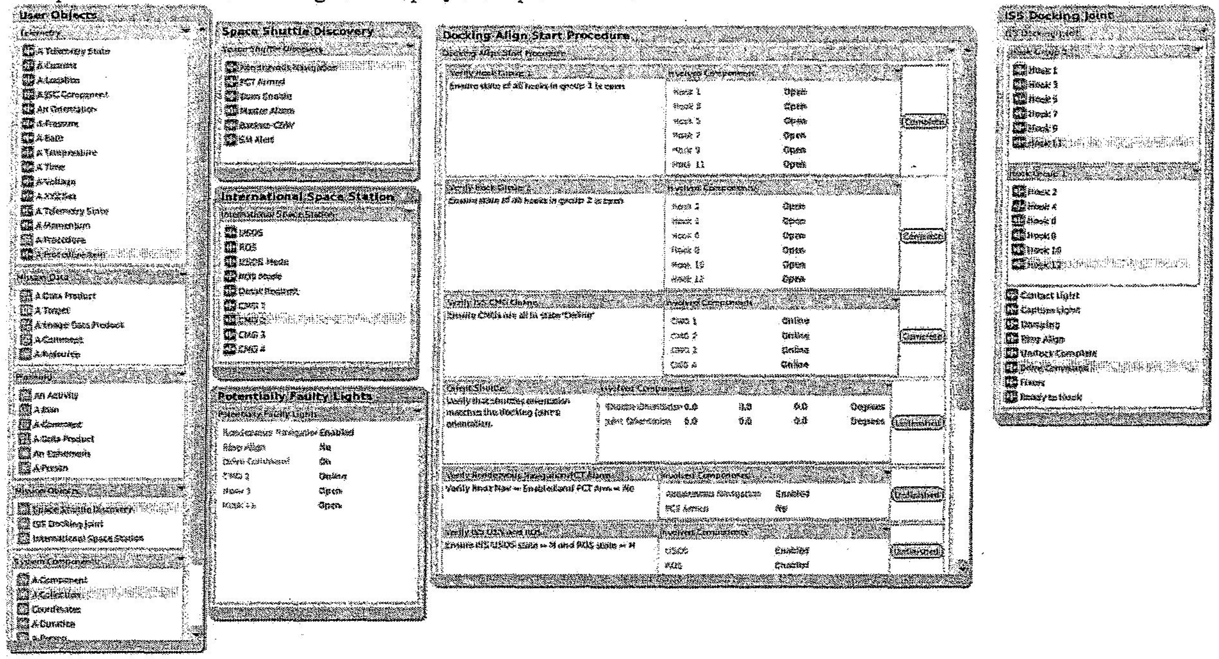


Figure 11. Telemetry and Procedures composition for ISS

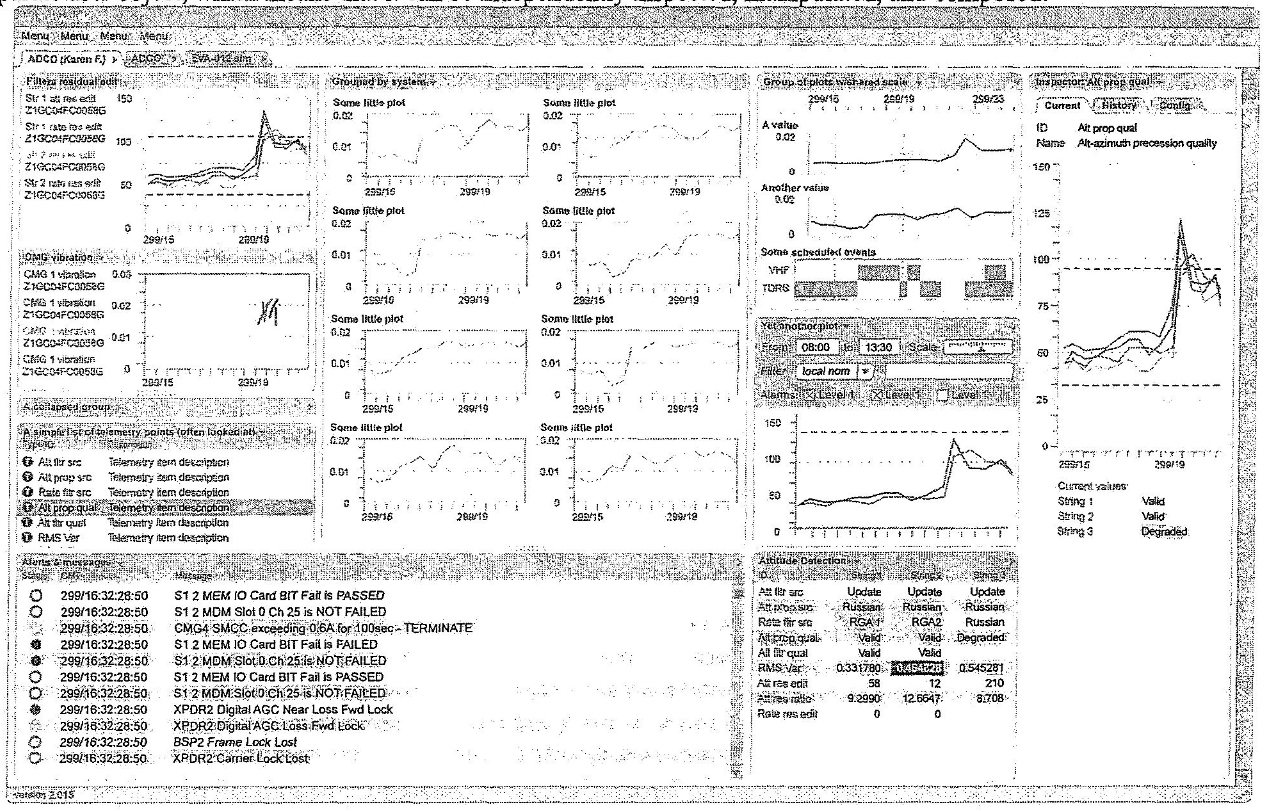
C. The Accountability Pilot

Figures 7 and 8 show the pilot application built and composed in 2005. Unlike the prototype, the pilot focused on depth, rather than breadth, and was required to connect to real data source. We chose data accountability, as that built on previous work conducted with the Jet Propulsion Laboratory (JPL), and provided an adaptable service and data source. The pilot successfully demonstrated the capability to build composable components on top of Eclipse, and it provided the basis upon which we are building the deployable infrastructure in 2006. Figure 9 shows the underlying representations on which the display in figure 9 is built. Each representation is a composable unit.

V. Scheduling and Telemetry Monitoring Pilot

In 2006, our focus is shifting toward supporting the vision for space exploration. We are maturing the frameworks towards a first deployment for developers and working with partners at JPL and the Johnson Space Center (JSC) on designing the first composable systems. Our domain focus for 2006 is telemetry and monitoring. We are building on our initial conceptual explorations, preliminary architecture, and initial component set to design and implement components for spacecraft telemetry monitoring. These will combine to create flexible monitoring "views" to be used by an initially limited subset of ISS mission control positions. These views will integrate a variety of data sources and types of information display, and demonstrate the use of components and composition for assembling both user-specific and position-specific views. The components themselves will permit user-driven composition and customization, ready exploration of data relationships, and highly flexible, on-the-fly combination of components to assist flight controllers in understanding telemetry values and analyzing the causes of events.

Error! Reference source not found. is a mockup illustrating the variety of components to be implemented, and a composition that might be created using them. Note that nearly every one of the elements in this user interface is a separate user object, which means that it can be independently inspected, manipulated, and composed.



A simple list of telemetry points (often looked at)

Symbol	Description
Alt fltr src	Telemetry item description
Alt prop src	Telemetry item description
Rate fltr src	Telemetry item description
Alt prop qual	Telemetry item description
Alt fltr qual	Telemetry item description
RMS Var	Telemetry item description

Alerts & messages

```
Status  GMT               Message
○       299/16:32:28:50   S1 2 MEM IO Card BIT Fail is PASSED
○       299/16:32:28:50   S1 2 MDM Slot 0 Ch 25 is NOT FAILED
        299/16:32:28:50   CMG4 SMCC exceeding 0.6A for 100sec - TERMINATE
●       299/16:32:28:50   S1 2 MEM IO Card BIT Fail is FAILED
●       299/16:32:28:50   S1 2 MDM Slot 0 Ch 25 is NOT FAILED
○       299/16:32:28:50   S1 2 MEM IO Card BIT Fail is PASSED
○       299/16:32:28:50   S1 2 MDM Slot 0 Ch 25 is NOT FAILED
●       299/16:32:28:50   XPDR2 Digital AGC Near Loss Fwd Lock
        299/16:32:28:50   XPDR2 Digital AGC Loss Fwd Lock
○       299/16:32:28:50   BSP2 Frame Lock Lost
○       299/16:32:28:50   XPDR2 Carrier Lock Lost
```

Attitude Detection

ID	String 1	String 2	String 3
Att fltr src	Update	Update	Update
Att prop src	Russian	Russian	Russian
Rate fltr src	RGA 1	RGA2	Russian
Att prop qual	Valid	Valid	Degraded
Att fltr qual	Valid	Valid	
RMS Var	0.331780	0.464528	0.545281
Att res edit	58	12	210
Att res ratio	9.2990	12.6647	8.708
Rate res edit	0	0	

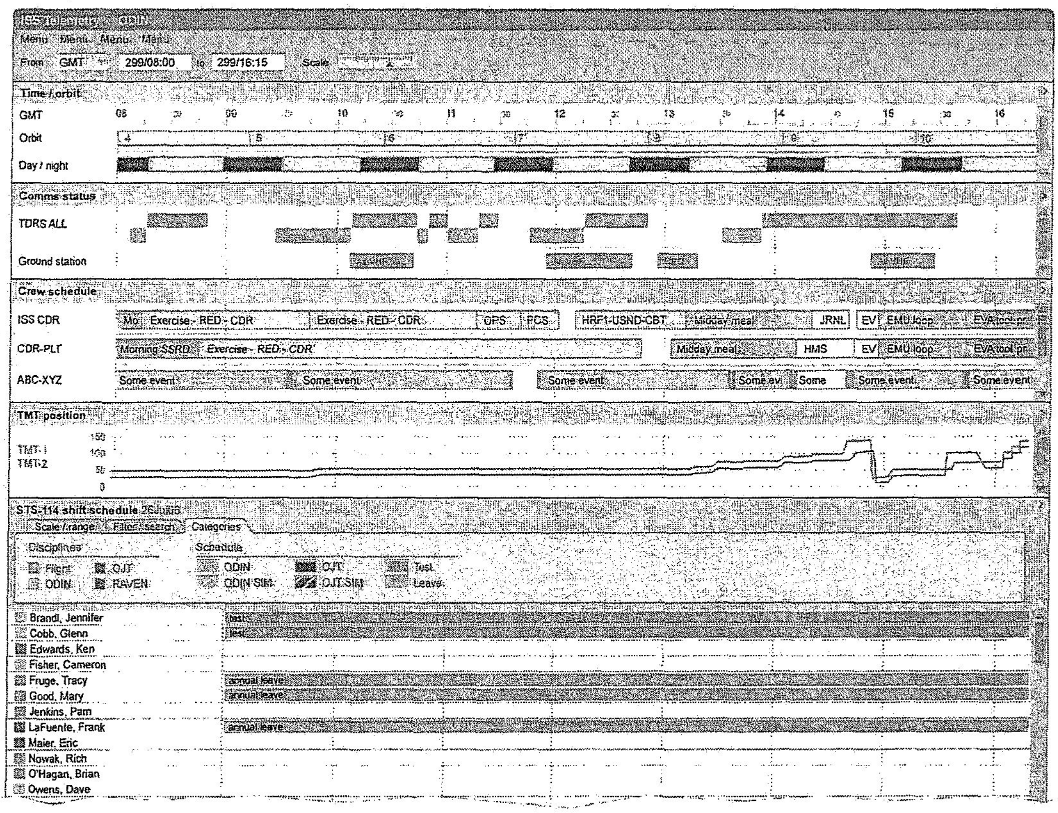
ID Alt prop qual
Name Alt-azimuth precession quality

Current values:

String 1	Valid
String 2	Valid
String 3	Degraded

Figure 12. Telemetry monitoring components

Error! Reference source not found. is a mockup showing a number of planning and scheduling components, also in development for FY06. Note that it includes a telemetry item whose value is plotted on the same time scale as are the schedules. Conversely, the previous mockup (Fig. 16) includes a schedule rendered according to the same time scale as the telemetry plot it is composed with. Both of these can be accomplished simply by dragging the items together. These examples illustrate the degree of dynamic component interoperability that MCT makes possible.



From GMT 299/08:00 to 299/16:15 Scale

Figure 13 Planning and Scheduling Components

VI. Conclusion

We have demonstrated the capability to build infrastructure and components to create composable systems. In 2006, working with our partners at JSC, Marshall Space Flight Center (MSFC), Goddard Space Flight Center (GSFC) and JPL, we are building composable systems in the planning domain (with JPL), and telemetry and monitoring (with JSC, MSFC, GSFC). We expect initial availability of the framework for developers to be at the end of 2006. At that point, we will be able to test the concept of assembly of ground systems from components developed by multiple organizations and multiple locations across many operational domains.