Integrating Engineering Data Systems for NASA Spaceflight Projects

Robert E. Carvalho
NASA Ames Research Center
Moffett Field, CA 94035
650-604-3593
Robert.E.Carvalho@nasa.gov

David G. Bell
USRA RIACS
Moffett Field, CA 94035
650-604-0771
David.G.Bell@nasa.gov

Irene Tollinger
NASA Ames Research Center
Moffett Field, CA 94035
650-604-5740
Irene.Tollinger@nasa.gov

Daniel C. Berrios
University of California Santa Cruz
Moffett Field, CA 94035
650-604-0470
Daniel.C.Berrios@nasa.gov

Abstract*—NASA has a large range of custom-built and commercial data systems to support spaceflight programs. Some of the systems are re-used by many programs and projects over time. Management and systems engineering processes require integration of data across many of these systems, a difficult problem given the widely diverse nature of system interfaces and data models. This paper describes an ongoing project to use a central data model with a web services architecture to support the integration and access of linked data across engineering functions for multiple NASA programs. The work involves the implementation of a web service-based middleware system called Data Aggregator to bring together data from a variety of systems to support space exploration. Data Aggregator includes a central data model registry for storing and managing links between the data in disparate systems. Initially developed for NASA's Constellation Program needs, Data Aggregator is currently being repurposed to support the International Space Station Program and new NASA projects with processes that involve significant aggregating and linking of data. This change in user needs led to development of a more streamlined data model registry for Data Aggregator in order to simplify adding new project application data as well as standardization of the Data Aggregator query syntax to facilitate cross-application querying by client applications. This paper documents the approach from a set of stand-alone engineering systems from which data are manually retrieved and integrated, to a web of engineering data systems from which the latest data are automatically retrieved and more quickly and accurately integrated. This paper includes the lessons learned through these efforts, including the design and development of a service-oriented architecture and the evolution of the data model registry approaches as the effort continues to evolve and adapt to support multiple NASA programs and priorities.

TABLE OF CONTENTS

1. INTRODUCTION

Spaceflight projects at the National Aeronautics and Space Administration (NASA) are governed by a broad set of standard procedural requirements [7]. These procedures span the project lifecycle from concept studies through technology development, system integration and test, launch, operations, and closeout.

To develop the capability for flight projects to readily implement these standard procedural requirements in a cost effective manner through the use of modern information systems, NASA chose to embed an information systems project in its flagship human spaceflight program – Constellation. By embedding the information systems project in the flight program, the project was able to readily obtain clear priorities and requirements for integrated systems, conduct user acceptance tests, and deploy systems to yield the benefits of the integrated systems. Benefits for the program were seen as critical for NASA as a whole, as the Constellation program represented ~19% of the overall NASA budget ($3.4B of $17.8B, based on actual costs in fiscal year 2009).

NASA's Constellation Program was charged with designing and developing the vehicles and systems to replace the Space Shuttle, return humans to the moon, and do so more cost-effectively over the mission lifecycle than previous human spaceflight efforts [10]. To support this goal, the Program needed to develop new systems and processes that would improve efficiency over the long term, and at the same time meet near-term schedule constraints.

One of NASA's goals was to limit the gap between the final Shuttle flight in 2011 and the first Constellation flight. Both near-term schedule and budget drove the need to reuse existing technologies. In terms of hardware, this included

reuse of the Solid Rocket Motors from Shuttle and the J-2 engine from Apollo's Saturn V rocket. For software, reuse included adoption of systems purchased or developed for previous programs. The Constellation Information Systems project developed an architecture for the future and a roadmap to achieve that vision. Central to this was an iteratively developed, service-oriented architecture to integrate engineering and operational data systems.

Data Aggregator (DAggr) was one of the first major steps in proving a future-focused architecture could support an environment of established legacy systems and new systems coming online over time. With DAggr in production since January 2010, this paper describes: 1) the problem NASA was working to solve, 2) the chosen implementation, 3) a discussion of the revisions made, 4) the challenges encountered, and 5) the benefits and lessons learned throughout the process with an eye toward helping other projects with similar needs for integrated engineering data systems.

2. CURRENT AND FUTURE STATES

Current State of Engineering Data Systems

The engineering data systems in different areas of the Constellation Program involved a mix of proprietary, open source, and NASA developed information systems. The proprietary systems included Cradle [4] for managing requirements and architectures, Windchill [13] as a Product Lifecycle Management tool, and Primavera [9] for schedule management. Other systems were NASA developed or NASA customizations of open source software such as the Integrated Risk Management Application (IRMA), the Problem Reporting and Corrective Action (PRACA) [8], and the Constellation Analysis Integration Tool (CAIT). Only after these tools were in widespread use was there an effort to bring these disparate information sources together to support the planned new ways of doing business.

Most of these applications provided their full functionality through a human user interface accessible via a web browser (e.g. Windchill, Primavera), and some provided limited functionality via their web interface. Most of these systems provided HTTP-based Application Programming Interfaces (APIs) using REST or SOAP, while a few only provided non-HTTP-based APIs. Each engineering data system was considered to be the authoritative source for records of one or more data types. For example, CAIT was considered to be the authoritative source for engineering analysis activities, and IRMA for risks.

Current State of Engineering Data

The engineering data were being copied between systems (used by different functional areas), resulting in multiple static copies of data. Each of these copies were stale to various extents, and therefore inconsistent with the authoritative source. For each copied data set, links were created that were not available in the authoritative source. For example, risks (from IRMA) were copied into the authoritative source for analyses (CAIT), and relationships were created to show that a particular analysis is being conducted to mitigate a particular risk. These links would not be available in IRMA.

In order to understand the current state of the entire Program, data has to be integrated from these various data source applications. This integration includes the links between the data items. The volume of data that is required to be integrated is significant. The first and second columns of Table 1 show the number of records for a sample of data types included in DAggr as of September 2010, and the third column shows the number of links between those records and data of another type.

Table 1. The quantity of selected records and links illustrates the large volume of data to be integrated by DAggr

Data Types	# Records	# Links
Requirements	61,197	Risks (199) Verifications (56,625)
Verifications	46,542	TVR (25,851)
Test Verification Requirements (TVR)	5435	Events (1365)
Risks	3904	Requirements (199)
Change Requests	943	Products (534) Documents (530)

With a large volume of data and links, trying to understand the current state of the program can prove daunting, even with the best of tools. With so many different tools, trying to manually bring together all the necessary data for integration is an expensive, time consuming, and error-prone task.

One user community, the Systems Integration Planning group, had a requirement to ensure that each Test Verification Requirement (TVR) (stored in Cradle) was checked against the associated Test Event (stored in Primavera). These checks served the purpose of giving the engineering community confidence that NASA understood the current state of the system (i.e., which TVRs have been performed, which are scheduled but have not occurred, and which ones are missing from the schedule). For example, a TVR that is missing an associated Test Event is a red flag against approval at milestone reviews and being confident in flight readiness assessments.

The Systems Integration Planners initially took a data import approach to capturing the links between TVRs

(stored in Cradle) and Test Events (stored in Primavera). All Test Events were imported into Cradle as a new data type and manually linked to the appropriate TVRs. After the initial import, the Test Event data in Cradle needed to be kept up to date via re-imports. However, upon re-import all the manually created links were invalidated because the Test Event IDs changed. Thus, each time the Test Events were re-imported into Cradle, the links had to be manually re-established. At that point, the user community started to track the relationships between TVRs and Test Events in MS Excel spreadsheets based on manually looking up the data in each of the authoritative systems (Cradle and Primavera).

These processes were time consuming both for the initial manual creation of the links (even in MS Excel) and subsequent updates. The data became stale quickly, and were prone to human error if an item was skipped or the wrong record happened to be pulled up in an authoritative source.

Other user communities had similar problems. For example, the Configuration Management community needed to link Change Requests (stored in the Change Request tracking system) to the Documents (stored in Windchill) that would be changed if the Request was approved.

In 2009 Constellation Information Systems had identified 14 different user communities looking to integrate data from more than 20 different data sources, with both of these numbers expected to grow as the Program progressed. It was recognized early on that neither the manual integration nor the copying of data would be sustainable for a Program that was intended to last for more than 20 years.

To understand the error rates in the copying approach better, imports between the risk system and the requirements system were analyzed at two points in time (March 2009 and April 2010).

In the March 2009 import, 182 Risk-Requirement links existed between 60 risks and 120 requirements. 25 of the 60 linked risks were not yet in the requirements system (42%) and 12 of the other 35 needed updates, resulting in 37 of 60 risks being updated (62%). In addition, three requirements that were linked to risks in the risk system did not exist in the requirements system. Metadata for 25 of the 120 requirements (21%) were found to be out of date in the risk system. 25 of the 182 links (14%) were missing and subsequently added in the requirements system (one for each of the newly created risks, with 23 of the links to one requirement, and the other two links to two other unique requirements).

In the April 2010 import (prior to the risk system being integrated with DAggr), there were 199 Risk-Requirement links between 76 risks and 130 requirements. 74 of the 76 copied risks (97%) were updated (52 involving one field only, and 22 (29%) involving other fields). In addition, 8 requirements that were linked to risks in the risk system did not exist in the requirements system. Metadata for 29 requirements of the 130 (22%) were found to be out of date in the risk system. No links were added in this import.

This analysis clearly shows that users of both tools were frequently attempting to integrate incomplete or outdated sets of data. The resulting integration had to be corrected and reviewed each time the data was transferred from one system to another. These results confirmed a clear need for an automated capability to integrate these data in order to reduce both the effort required and the rate of error in results.

Future State Goal

The desired future state was an engineering environment where data is automatically integrated directly from the authoritative sources, with uniform authentication and access, and common interfaces. The goal of this future state was to: 1) improve data accuracy, 2) improve data accessibility over the full program lifecycle, and 3) provide efficiencies in data management. To support this future state, the Constellation Information Systems project focused on implementing a Service Oriented Architecture (SOA), including semantic information in the style of the semantic web [11]. This would support adaptive machine understanding and improved navigation and search of data.

In this future state, use of non-authoritative copies of data would be minimized through automated data aggregation, and consequently rates of engineering data discrepancy would also be reduced wherever automated data aggregation was used. User communities could continue to manage specific data in tools explicitly designed for the management of that data. Other user communities would also be able to implement their specific processes with tools designed for them, but using aggregated data direct from the authoritative sources.

In addition to improved decision making in engineering, this future state would also provide benefits to operations by enabling integrated access to engineering data in applications optimized for processes associated with operations. One of the central goals of the Constellation program was the need to reduce operational costs for the integrated system as compared to Space Shuttle and International Space Station programs. Within those programs, much of the data required for operations was compartmentalized into various systems, and quite often data were manually copied from one system to another, or managed through paper documents. Creating accurate associations between such data could be greatly enhanced if links were restricted to only authoritative source data.

Constellation planned to improve the situation by making all essential data and the links between the data available to operators. An example of the utility of this is when an operator is attempting to address an anomaly in flight, they often need to refer to the history of a particular part, the

various tests it went through, the problem reports and waivers associated, and the design of the part [6].

At the same time, NASA as a whole was moving to consolidate the authentication approaches across all agency tools, so that users would have only one username and password to manage, rather than one for each system. This plan was incorporated into the information systems roadmap.

As much of the Constellation data was sensitive, access would have to be controlled and tracked. Initially, each of the tools had their own account management approaches, and users had to get separate approvals to access each one. Any integration of the data would need to provide the same levels of access and data protection as was available in the original data source.

3. Implementation Approach

The core of our approach was to enable automated integration of authoritative engineering data via development of the DAggr System. DAggr was intended to be developed in a rapid fashion, using existing capabilities where possible, and to have minimal impact on the existing data systems or processes. DAggr was defined as a core architectural capability to meet the fundamental user need to access and integrate authoritative data regardless of the system the user or data is in. The fundamental functionality objectives for DAggr are:

- Accessing data from authoritative sources
- Querying to find the necessary data
- Linking data together and managing these links as authoritative data
- Retrieving and combining selected data for reporting and analysis

The initial drive to get DAggr operational was to support the Systems Integration Planning for Constellation's Preliminary Design Review. This provided a total development time for DAggr of only four months to production release. The initial release was accomplished with twenty person-months of work on the core DAggr system.

DAggr Architecture

The DAggr system was composed of several layers, as seen in Figure 1. The core system was composed of a set of data source adapters to provide standardized interfaces for retrieving data, a link registry to manage the cross-system links, and the core DAggr system to decompose the requests and compose the responses.

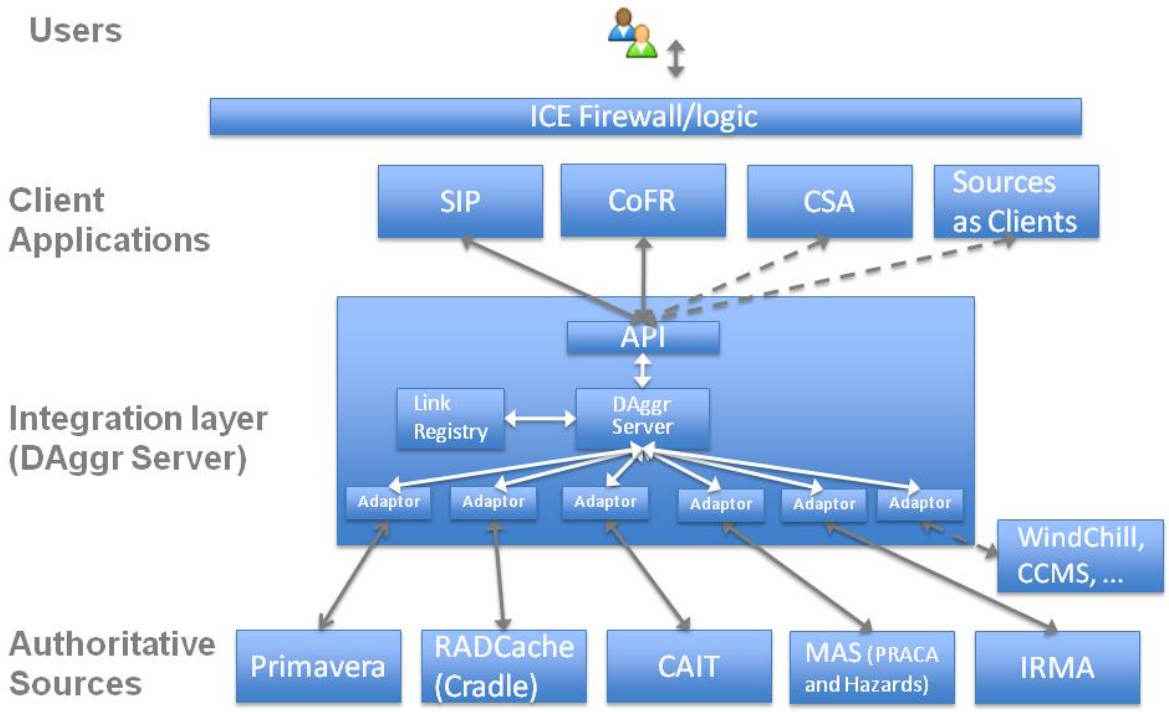

Figure 1 -- Architecture showing the various components of the DAggr system

The team decided to standardize the following aspects of the services: 1) approach for retrieving authoritative data, 2) approach for managing links between data, and 3) approach for controlling data access.

For retrieving authoritative data, the approach chosen was to provide two main services: 1) discovery service to identify available source systems and queries allowed with those source systems, and 2) query interface to retrieve authoritative data from one or more sources. The discovery service enables integrated applications to dynamically adapt to an evolving set of available services. The query service provides a standard interface for browsing and querying authoritative sources (e.g., field-value pair queries), and retrieving cross-system links stored in the external link service and/or the internal links available in the source system. The standard interface was enabled with a modular set of adapters to authoritative source systems. Adding a new source system requires authoring a new adapter that maps the standard API to the source-specific API, and registering the source system in the link management service.

This approach was implemented using HTTP with Uniform-Resource Identifiers (URIs) in a RESTful manner. This approach was chosen for the ease by which application developers could learn and use it, including the ability to simply use URLs in a web browser to make requests to the services and view sample responses. A list of sample requests was the most commonly used documentation by developers integrating applications with the web services.

For managing links between data, the approach chosen was to establish a link management service external to other source systems, as the authoritative source for two data types: 1) source system identification for each data type, and 2) links between data in two different source systems. The link management service provides standard functionality including link Create, Read, Update & Delete (CRUD) services, link access controls for individuals and groups, version history, and a link approval workflow. The endpoints of each authoritative link include unique identifiers to the source system and to the specific record being linked to in the source system. Additionally, any approved user of the service can create personal links to any data items they can access. A configurable workflow allows personal links to be promoted for use by all approved users.

Cross-system links are required for various work processes and required reports. Two types of reports that were analyzed include those produced for Systems Engineering referred to as System Integration Planning (SIP) reports, and those produced for Configuration Management referred to as Configuration Status Accounting (CSA) reports. As an example, one of the SIP reports required integration of 10 unique data types (e.g., requirements, verifications, test verification requirements, and events) with two to five fields each, from two separate authoritative data sources (e.g., requirements, verifications and test verification requirements from one source, and events from a second source). As another example, one of the CSA reports required integration of three unique data types (i.e., change requests, products, and documents) with one to eleven fields each, from two separate authoritative data sources (i.e., change requests from one source, and products and documents from another source). For these example reports, four required link types were cross-system links (TVR-Event, CR-Product, CR-Document).

The choice was made initially to implement the link management with a semantic repository. This would provide benefits as described in [5]: using semantic knowledge to integrate and link the data, and providing a standardized representation of all data and links. Further benefits of the semantic approach are discussed in section 5. The data source specific data models were mapped to the ontology within the DAggr server to provide a common framework. A selection of data types was retrieved from the data sources and populated into the ontology used by the link registry. Cross-system links were also stored in the ontology. The ontology also served to provide a standardized XML data structure to deliver all data to the clients, including the cross-system links.

Authentication of users is handled via a single Lightweight Directory Access Protocol (LDAP) directory service, initially with all applications behind a single Apache HTTP server, and subsequently with all applications behind their own HTTP servers using the NASA eAuth approach. A group of users is managed in an LDAP directory service, and only members of that group are authorized to access the DAggr web services or clients. To access the clients, the users log in through Apache, which checked their membership in the group, and if they were a member then Apache passed their credentials to the requested DAggr client application (e.g. SIP), which would then pass their credentials to DAggr web services and on down to each of the data sources being accessed.

Users who had an account in an authoritative source were given access to whatever their account was authorized to see in the authoritative source. Users who did not have accounts but were approved DAggr users were given access only to baseline data. The specification of baseline data was left to the data owners within each data source. This policy ensured that the users were only allowed to see the same data they would be able to see in the data sources directly, or baseline data if they did not have an account in the source system.

In addition to the core DAggr components, a cache was developed as a work-around for significant performance and API limitations of the key authoritative source for requirements data. The cache was automatically loaded with an updated copy of all data from this source every night, and made accessible to users via a RESTful web service integration with DAggr. The caching rate was set to nightly for two reasons: 1) this would not impact server load during primary usage times, and 2) any given data item

in the system did not change so rapidly that a more frequent update would be required.

Initial DAggr Clients

DAggr is a service-oriented middleware. Each client was provided the same set of services. These were:

- Get available (sources, data types, fields, links),
- Multi-system search (decomposed by DAggr to search each data source),
- Structured Query (query by source, data type, field, or link),
- Get item details,
- Get linked items,
- Create Link,
- Approve Link
- Get data source status (up or down)

Through the development cycles, the DAggr team developed or supported the development of various DAggr client applications. Each is described below.

Systems Integration Planning

The first client application developed using DAggr services was the Systems Integration Planning tool (SIP). The primary function of this tool was to provide the information and links to ensure that all space vehicle requirement verification activities were appropriately planned for. SIP users would search for the TVRs and link them to the appropriate schedule events to make sure the plan included all required verifications. The data sources integrated for this tool were Cradle (providing requirements, verification requirements, and verification objectives), Primavera (providing schedules and schedule events), and CAIT (providing analysis and integration activity descriptions).

The SIP user community also wanted to extend the capabilities of the tool to search and link data from other sources, including part information (from Windchill), risk data (from IRMA), problem reports (from PRACA), and test result data from a system yet to be selected. SIP was deliberately developed so that as new data sources and types were added to DAggr, they would be available to the user without changes to the client application code. As the first DAggr client, SIP served to demonstrate and test the accuracy and efficiency of data aggregation.

Certification of Flight Readiness

The second client application developed using DAggr services was the Certification of Flight Readiness (CoFR) Dashboard. The primary function of this tool was to provide mission managers and system integrators the capability to monitor the overall progress of a mission or test flight to launch. The goal was to allow monitoring at a high level, and then the capability to drill down where issues were identified, to get the details. The primary users of this system would take advantage of the links created by other users earlier in the design cycle. The data sources integrated for this tool were Cradle (verification status), Primavera (schedule from now to launch), CAIT (analyses to be completed before launch), IRMA (risks identified for this mission), PRACA (problem reports for this mission's hardware and their status), and a CoFR specific system to track the actions assigned at reviews until they were closed. The future desired data would include a broad range of program data, from training status for crew and flight controllers to test and inspection results. This client also served as a demonstration of the extensibility of the DAggr architecture previously proven in the SIP release. The entire development for the CoFR release, including adding the additional data sources and types to DAggr, took only two months, with four person-months worth of work on the core DAggr system.

Configuration Status Accounting

The next client application developed using DAggr services was a tool for Configuration Status Accounting Reports. The primary function of this tool was to support the tracking of the integrated baseline of the Constellation Program. It is important when conducting analyses to ensure that a consistent set of data is being used. To do this with so large a parallel effort required the capability to know which version of the Orion design and which version of the Ares I design were being used, all the way down to the atomic level of components and requirements. This client allowed the users to link all of the appropriate data items with each Change Request as it went through the system, and then get the proper approvals for those links. The data sources integrated for this tool were IRMA, PRACA, and CAIT, plus Windchill for documents and product structure information, and the configuration data management tool suite for tracking the approval of documents and products.

Data Sources as Clients

Several user communities who directly used one or more data sources also expressed interest in receiving data from the DAggr system. For example, the user community responsible for Hazard Reports wanted to be able to make links to related TVRs and parts within their application. Since Hazards was a NASA modified open source application, it was possible to embed a DAggr client. This would enable them to stop the copying of data from other systems, and would also enable them to create, approve, and display the links between their data and the data in other systems. The PRACA, IRMA, and CAIT tools all proved capable of showing links and data from DAggr with two to four person-months worth of work. The users of the Cradle tool were also interested in this capability, but as it was a COTS (Commercial Off-The-Shelf) tool, this effort was postponed until it could be incorporated by the vendor at a later release.

4. RE-IMPLEMENTATION AND EXTENSION

With the termination of NASA's Constellation Program, the driving forces behind the continued development of the

DAggr changed. Instead of targeting data linkage capabilities for Constellation Program managers and analysts, the system was redesigned and re-purposed for integration of data from existing ISS applications and new NASA human spaceflight programs and projects with substantial data linking needs. The shifts in targeted systems and users required a more streamlined data model registry for Data Aggregator in order to simplify adding these systems' new data. We also required a unified query specification for Data Aggregator to support cross-application querying by client applications.

Simplified Data Model Registry

As we discussed earlier, the initial implementation of the system required that source data descriptions be modeled in a semantic repository (AllegroGraph). Configuring, maintaining and extending the repository proved to be laborious and unwieldy. We decided to remove the semantic repository and build our own repository for this information using a common RDBMS (MySQL), so that new data sources and their objects could be more quickly and easily added.

New Data Sources

The demonstrated ability of the DAggr middleware to support integrated views on data from multiple Constellation Program systems stimulated interest in similar capabilities for systems used by the International Space Station Program. The program identified a specific need to coordinate records from the PRACA, Vehicle Master Database (VMDB), and Drawing Access and Retrieval Tool (DART) systems. The PRACA system contains over 26,000 records related to identifying, analyzing and tracking ISS part issues. The VMDB and DART systems contain metadata and drawings for all ISS Program parts, including physical dimensions, manufacturer information and CAD drawings. Users of PRACA manually enter information such as part number, drawing number, and manufacturer name, without automated support for looking up this information in the systems where it is stored, DART and VMDB. This leads to occasional errors of transcription of the information as it was copied between the systems.

We developed data source wrappers for VMDB and DART to support automated access and integration of the part data from these two systems as users create PRACA records. The wrappers called Oracle PL/SQL scripts to look up part, manufacturer, and drawing information in the databases that VMDB and DART access, and transform the information into DAggr items, which can be delivered to clients such as the PRACA system. We similarly "wrapped" the PRACA system so that users could create DAggr links to VMDB and DART records when creating or updating PRACA records (Figure 2). This dynamic, semi-automated linking process ensures that part information from the VMDB and DART sources is incorporated into PRACA records without the errors that can happen through manual transcription.

Unified Query Specification

The original design of DAggr required that clients use the legal syntax of queries accepted by each data source. Queries submitted to DAggr were merely re-transmitted to data sources unchanged, and legal query syntax was wholly

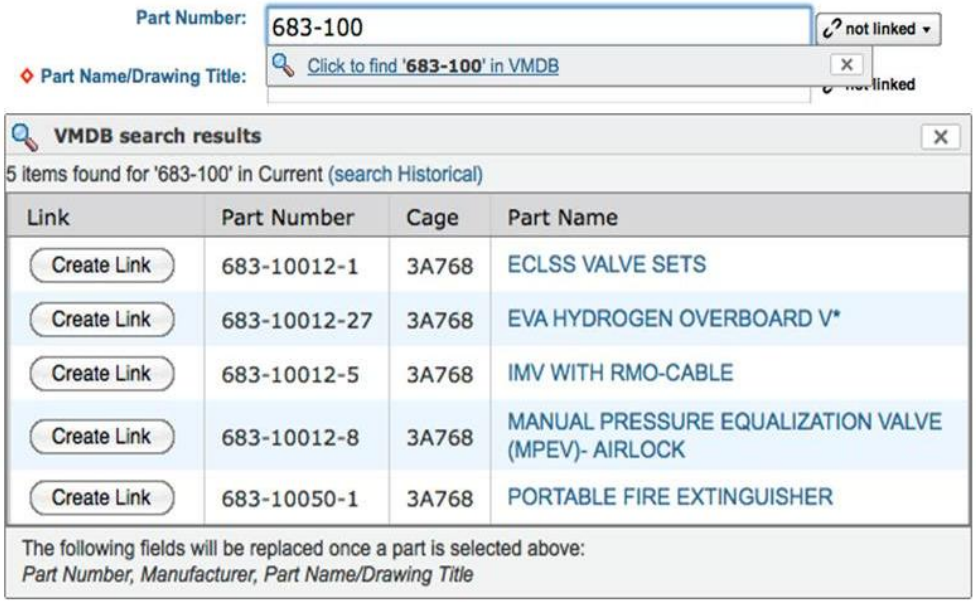

Figure 2 -- VMDB Search within PRACA Interface

dictated by the data sources. Thus, if an application using data from sources A, B and C wanted to query each data source through DAggr, it had to use 3 different query syntaxes. In order reduce this burden on clients, we chose to develop and implement a single DAggr query syntax. We selected SQL92 as its basis, with DAggr item type names replacing data source and table names, and DAggr item type attribute names replacing table column names in the syntax. For example:

SELECT * FROM VMDB_PART

WHERE PART_NUMBER LIKE '683-100%'

5. CHALLENGES REMAINING

Although DAggr's early releases are in production, there remain several significant challenges ahead. This section discusses three categories of challenges and possible solutions that the DAggr team considered for each.

Version Control

Each of the data sources which were brought into DAggr had a different approach to version control. Some handled versioning at a larger granularity (e.g. an entire hazard report), while others handled it very atomically (down to the field level). Additionally, some changes required a formal review and re-approval, while others did not. Users of the various DAggr clients, and other future users also had their own ideas about what level of versioning they were interested in. In some cases, users were also interested in version-sensitive links between items. For example, Flight Software Load 3.6 would meet version 8 of a requirement. Version 8 of that requirement was generated based on a particular PRACA which occurred in Flight Software Load 3.5. Users were also interested in using the version-sensitive links to monitor changes. So if a particular Hazard was addressed by version 3 of a requirement, then before version 4 of that requirement could be approved, the Hazard would need to be checked.

While the ontology would have no issue in tracking all these variations of versions, the issue would be in standardizing the version information provided by the data sources, and in determining a reasonable process for the various approvals. While one could theoretically impose a standard for versioning on most data sources, inclusion of COTS and legacy systems can make such an approach very expensive or even impossible. One alternative considered was to have the registry track all the versions of the data items, but this would require either significant polling or moving to a push model instead of a pull model which may impact performance. This option was considered especially valuable for certain user communities, as they had requirements not only to see the current state, but also to see the state of the system at a particular point in time. For example the CoFR client needs to be able to show the knowledge of the system at the time the Certification of Flight Readiness was issued in case there was a question later.

As Constellation was in the process of moving from a document-centered approach to change management to a data-centered approach, one of the key issues was what level of item and what level of data change would trigger formal reviews. Would each link between data items need to be formally approved? The approach currently taken in DAggr varies depending on the kind of link involved, but generally follows the process where any user can create a link, but only that user can see it until it is submitted to be a Program Baseline link. Once approved, all users would be able to see it. But with thousands of requirements, schedule events, and parts, even just the full set of links for SIP would be in the tens of thousands. What kind of review process should be in place for that volume of data? This issue remains unresolved.

Performance Limitations

Since DAggr is a system that pulls data directly from authoritative sources distributed across the country, the time for it to respond to a request depends on the time it takes for the authoritative sources to respond to requests, plus the network transfer time, plus processing time. Depending on which sources are being queried in a request, the first page of data in SIP comes up in as quickly as two seconds, with all linked data showing up progressively over the next five seconds. Linked data includes both cross-system links stored in the link registry, as well as internal links stored in the authoritative source. However, some of the data sources were already approaching performance limits not only for remote calls from the aggregation service, but also for their direct users of their native user interface. If the user base of DAggr expands, this may impact not only DAggr client users, but the users of the source systems as well.

DAggr users were also interested in a broader search capability, including full-text search of all data items and fields, relevancy ranking, and the ability to sort results based on particular criteria. Some of the sources did not provide services for full-text searches. Some could only provide back the first *N* results (with a maximum cap on *N*), which would interfere with retrieving the full set to sort. In other cases, the system might not be able to support such searches from thousands of users.

To address these performance issues, caching data from the sources when it was requested was one option considered. This would speed response time for common requests and searches. However, it would be of limited value for the more full-featured searches as the search would still have to be sent to each data source, to catch new and changed data items.

Another approach considered was that of a Data Warehouse (see cache discussion above), holding cached copies of all the data in the sources. This would also then support

common search indexing approaches. Two different approaches to warehousing were considered.

The first was to move to a push model rather than a pull model, so that any new or changed data in the data sources would be reported to the Registry. However, this would have required more extensive modifications of the data sources to get them to push the data, and would have run into trouble with their varying version control approaches (since it is not clear which changes need to be pushed).

The other approach considered was to move to a polling approach, where the Registry would periodically ask each data source "What do you have that is new or changed?" This would have required less modification of the sources, but some sources would not have been able to answer that consistently due to different approaches to versioning, so the poll would shift to a full data pull every interval. This would have imposed a load on the networks and the data sources, which was considered unacceptable in some cases.

Authentication and Control

Not all of the data the users were interested in accessing through DAggr was stored on NASA systems. In some cases, NASA vendors were the primary source for information. The solution of a NASA single sign-on would not help in the case of such remote systems. Getting access to the data in those systems would have required much more extensive negotiations, as well as figuring out a way to either share user identity information or to map from the NASA system to the vendor's system. To simplify this in the near term, DAggr focused on discrete transfers of data. On a periodic basis, the vendor would provide a copy of their data into a NASA system which would then provide the services which DAggr needed.

Although NASA has begun migrating its systems to a single approach to authenticating users, users must still request access to each system individually, or get only baseline level access under the current approach. The user communities were asking why all System Integrators didn't have access to all the same data. One approach considered would use role-based access controls. However, in order to do a role-based access, DAggr would need some way to map the users to their various roles (as NASA employees often have multiple functions). Initial efforts to build a tool for this were begun, but the key effort in setting this up would be to provide some way for the user community at large to maintain this data. Once such a system achieved an appropriate level of usage, then all data sources would need to call on the services of such a role-mapper so that those roles would be available internal to the tool. Then the permissions within each tool would need to be reset to use the roles, rather than the tools' internal groups and users. This might prove difficult with COTS and legacy tools.

Lastly, while users were very happy to begin viewing integrated data in their particular tool of interest, there was a general consensus that eventually the users would need the capability to modify this data, and have those changes pushed back to the original source system. A flight controller sitting on console in mission control should not only be able to review all the past Problem Reports (PRs) on a part, but should be able to create and file a new PR, and have that go directly into PRACA. A next step would be to move beyond just supporting Read permissions to supporting all CRUD permissions. However, with the current baseline user approach, there were strong concerns that the traceability and control on such changes would not be adequate. Moving to a role-based level of access would improve this, but the system would still need to be set up to show which flight controller filed that PR. Moving to a full CRUD service model would also have required more extensive development on the data sources to provide the additional services. It was hoped that as industry moves more generally to a service-based model that COTS tools would provide this capability, but imposing it on legacy systems would still prove a challenge.

6. BENEFITS

Our experience with DAggr revealed several benefits of this initial implementation of a SOA. This section will summarize the two most significant benefits.

Improved Business Intelligence

One of the primary benefits of the DAggr system is access to better business intelligence. Business intelligence is defined in [14] as "applications and technologies which are used to gather, provide access to and analyze data and information about an enterprise, in order to help make better informed decisions."

At the most basic level, DAggr eliminates potential discrepancies in data that get introduced when data is manually copied from the authoritative source to other sources and to reports and presentations. As described earlier in this paper, discrepancies between the authoritative data in one system and a copy of that data in another system were observed to range from 14% to 97% for a range of data. Automated data aggregation on demand eliminates these discrepancies, allowing decisions to be made on the authoritative data rather than an incorrect copy of the data. Discrepancy rates observed in this DAggr effort are comparable with another study where 40% discrepancy rates were observed [1].

In addition, by automating data aggregation, engineers and managers can spend more time on analyzing data rather than just aggregating it, and can enable more standardized formatting of data. This can make processes more repeatable and improve human interpretation of the data.

As one example, systems engineers indicated in interviews that they spent up to half of their time gathering and assembling data from a variety of tools, and therefore had less time to make the critical decisions they were tasked

with. The links between the data were essential to providing an integrated view. In the SIP application, the primary function was to ensure that each TVR was scheduled. Since this data lived in two different places prior to DAggr, system integrators were forced to extract it all and use Microsoft Excel to show the relationships between the data. This led to challenges both in accuracy of the copying (as discussed in section 2) and in keeping the link data up to date so that the whole team could understand the current state. DAggr's ability to integrate the data within one application and store the relationships created by the system integrators reduced these risks.

As a second example, under the existing practices, a Certification of Flight Readiness was only signed after months of tiered Flight Readiness Reviews (FRRs). For each review, Microsoft PowerPoint charts would be prepared manually summarizing the data gathered from a variety of sources, including the PowerPoint charts prepared for lower tier reviews. By the time the PowerPoint charts reached the top FRR, the data shown on them could be stale. By providing a CoFR client with direct access to the original data sources, a mission manager need not wait months to find out the status of the mission, but could check at any time. This also improves the repeatability of the process, so that the data seen does not vary depending on the choices of the PowerPoint editors at several steps along the way.

Support for Multiple User Communities

A key benefit of using an abstraction model such as the ontology in the DAggr server is that it greatly facilitates expansion of the system. The clients base their data model on the DAggr server data model. Thus, when new data sources are added, the client applications can incorporate that data with no modification or recompilation. Even the DAggr server did not require significant modifications to incorporate the new data sources. The changes were primarily adding a new adapter and updating the ontology. The adapter development was fairly straightforward, as the full list of calls for each data source was standardized and short. This ease of adding data sources was demonstrated in our second (CoFR) release, which took less than four person-months to add three data sources.

Another benefit, as discussed in [14] is the ability to support access to the data by new clients/tools with little modification to the server and none to the data sources. Different users will need to perform different tasks with the data, and want to have their own tools to do this in. While the SIP client developed was easily modified to function as the CSA client, with the CoFR release it was also demonstrated that an entirely new client could be added with very little work on the server side, in a very short amount of time.

7. LESSONS LEARNED

This section will describe the lessons learned from the DAggr experience that should have a general applicability for organizations seeking to move toward an SOA.

Complete the aggregation cycle quickly

The DAggr development experience shows that it is possible to provide benefits to an organization quickly using a SOA without much expense. With twenty person-months of development effort in only four months time, three data sources and one client application were integrated in a production environment. Following that initial milestone, roughly one new data source each month was added for a current total of ten adapters to different authoritative data sources, with sets of adapters deployed to production on a quarterly basis.

One key step in doing this so quickly was the parallel development, where the adapters, the DAggr server, and the client were all developed concurrently. This was greatly facilitated by a simple interface definition in REST, which also made testing easier, as any developer could use a web browser to test the various calls being made. This parallel development effort also required multiple development and integration environments, so that each layer had a stable version of the next to test against while the other layers were making rapid changes.

This parallel development was also facilitated by implementing with the basics of what every developer has on their development machine – a programming language and an integrated development environment. Attempts at using other approaches that involved tools marketed as being facilitators of SOA implementation proved to be inhibitors. Developers who wanted to participate needed to procure a license for the tool and get specialized training to learn how to use it. By implementing without such tools, developers readily transitioned into the role of adapter developer without requiring procurement of tools or specialized training.

Develop incremental capabilities for specific applications

Organizations with established tools and processes can be resistant to migrating to an SOA [2]. In order to avoid some of the difficulties here, the DAggr development effort made some conscious decisions to focus on developing incremental capabilities for specific end user applications. This included the decision to start with read-only services to minimize impacts on legacy and COTS tools. Another choice was to start with a small but critical user community to prove the capability quickly and prove initial benefits. Having this set of users enthusiastic about the work then made it easier to move into other user communities and add more data sources as the momentum built. It is important to get the whole organization comfortable through small, quick steps.

Be flexible and adaptable

DAggr incorporated data sources that span the range from proprietary COTS to open-source to in-house legacy tools for a range of applications. The interfaces for these data sources ranged from sources that didn't have any usable http-based interface to ones that had RESTful interfaces. To integrate across the range of these sources for a range of applications, being flexible and adaptable was key. Incorporating data sources requires use of a variety of flexible solutions to adapting them [3].

In-house developed tools with active development teams provided the easiest solution, by adding components necessary to support the defined interface services. COTS and legacy tools can be more difficult, depending on what services or interfaces are available, and the overall performance needs of those systems [12]. In one case we were able to wrap the services of a modern COTS tool. In another, for performance reasons, we wound up using a caching solution to protect the original COTS source and meet our functionality needs.

8. SUMMARY

Moving to a web of integrated engineering data systems is a viable option today, using an incremental SOA approach. This approach enables automated integration of linked engineering data, which eliminates discrepancies between copies of linked data that were observed to impact from 14 to 97% of the copied data items due to manual copying. This also enables reduction in lifecycle costs associated with manual copying and linking of data. A data aggregation service can readily be tailored to specific user community procedural requirements, yielding near-term benefits tailored to needs of engineering organizations. The modularity and flexibility of the approach is expected to enable long-term benefits as well.

Future large endeavours with large legacy systems and processes must take an iterative approach in migrating to SOAs to avoid significant disruptions. Flexible, modular SOA migrations can be an important step for such efforts to address the technical and organizational aspects of the migration.

REFERENCES

[1] D. G. Bell, D. M. Maluf, Y. Gawdiak, P. Putz, K. Swanson, "The NASA Program Management Tool: A New Vision in Business Intelligence" *IEEE Aerospace Conference 2006*, March 2006

[2] N. Bieberstein, S. Bose, L. Walker, A. Lynch, "Impact of service-oriented architecture on enterprise systems, organizational structures, and individuals", *IBM Systems Journal* 44(4):691-708, 2005.

[3] G. Canfora, A. R. Fasolino, G. Frattolillo, P. Tramontana, "A wrapping approach for migrating legacy system interactive functionalities to Service Oriented Architectures", *The Journal of Systems and Software* 81:463-480, 2008.

[4] "Cradle Overview" in http://www.threesl.com/pages/Cradle/English/Content/Products/overview.php, accessed October 2011.

[5] D. Fils, C. Cervato, J. Reed, P. Diver, X. Tang, G. Bohling, D. Greer, "CHRONOS architecture: Experiences with an open-source services-oriented architecture for geoinformatics", *Computers and Geosciences* 35:774-782, 2009.

[6] *NASA Procedural Requirements for Mishap and Close Call Reporting, Investigating, and Recordkeeping*, NPR 8621.1B 2010.

[7] *NASA Procedural Requirements for NASA Systems Engineering Processes and Requirements*, NPR 7123.1A 2009.

[8] C. B. Green, I. V. Tollinger, C. D. Ratterman, G. . Pyrzak, A. A. Eiser, L. . Castro, & A. H. Vera (2009) "Leveraging open source software in the design and development process", *In K. Hinckley, M. R. Morris, S. Hudson, and S. Greenberg (Eds.): Extended Abstracts of the 2009 Conference on Human Factors in Computing Systems (CHI 2009)*, (pp. 3061-3074). San Jose, CA: ACM 2009.

[9] "Primavera Scheduling Software" in http://www.ipsys-3.com/primavera_scheduling.html, accessed October 2011.

[10] J. L. Rhatigan, J. M. Hanley, M. S. Geyer, *Formulation of NASA's Constellation Program*, NASA Special Publication SP-2007-563.

[11] N. Shadbolt, W. Hall, T. Berners-Lee, "The Semantic Web Revisited", *IEEE Intelligent Systems* 21(3):96-101, Jan-Feb 2006.

[12] X. Wang, S. X. K. Hu, E. Haq, H. Garton, "Integrating Legacy Systems within the Service Oriented Architecture", *IEEE Power Engineering Society General Meeting 2007*:1-7, 24-28, June 2007.

[13] "Windchill Business Process Management" in http://www.ptc.com/products/windchill/, accessed October 2011.

[14] L. Wu, G. Barash, C. Bartolini, "A Service-oriented Architecture for Business Intelligence", *IEEE Service-Oriented Computing and Applications 2007*:279-285, June 2007.

BIOGRAPHIES

Robert Carvalho *is a Systems Engineer with NASA Ames Research Center's Intelligent Systems Division. For more than 15 years, he has supported a variety of NASA projects. He is currently working on the ground and mission operations systems for the Interface Region Imaging Spectrograph mission. His research interests include model based systems engineering and knowledge management for safety and design. He received his B.S. in Computer Engineering from Santa Clara University.*

David Bell *is Director and Senior Scientist at the USRA Research Institute for Advanced Computer Science, located at the NASA Ames Research Center. Prior to working at NASA, David worked for ten years at the Xerox Palo Alto Research Center, and previously held an appointment at MIT where he led a research program in the Center for Innovation in Product Development. David is co-inventor of multiple patent and patent-pending information system technologies, including extensible blog technology called Sparrow Web, public and private electronic markets, and the NASA Program Management Tool; as well as co-inventor of an early crowdsourcing technology called Eureka. David received his Ph.D. from Cornell University with a dissertation on the dynamics of product development processes.*

Irene Tollinger *leads the Human-Computer Interaction (HCI) team in the Human Systems Integration Divsion at NASA Ames Research Center. She has won numerous awards for her work in support of NASA missions, including Constellation and the Mars Exploration Rovers. She graduated from the Human Computer Interaction Institute at Carnegie Mellon University in 2002.*

Daniel Berrios *has over 15 years experience in the fields of scientific computing and informatics. He has led multi-center research projects in epidemiology while at the University of California, and managed scientific and technical information retrieval research while at Stanford. He currently develops collaborative information systems for scientists, technologists, and engineers at the University of California, Santa Cruz. He has a bachelor's degree in mathematics from Brown University, a doctorate of medicine from the University of California, San Francisco, a master's certificate in public health from the University of California at Berkeley, and a Ph.D. in biomedical information sciences from Stanford University*