DATA DRIVEN CAMPAIGN MANAGEMENT AT THE NATIONAL IGNITION FACILITY

Douglas Speck, Bruce Conrad, Steven Hahn, Paul Reisdorf, Scott Reisdorf, LLNL, Livermore, CA 94550, USA

Abstract

The Campaign Management Tool (CMT) Suite provides tools for establishing the experimental goals, conducting reviews and approvals, and ensuring readiness for a National Ignition Facility (NIF) experiment. Over the last two years, CMT has significantly increased the number of diagnostics that it supports to approximately 40. Meeting this ever increasing demand for new functionality has resulted in a design whereby more and more of the functionality can be specified in data rather than coded directly in Java. To do this support tools have been written that manage various aspects of the data and to also handle potential inconsistencies that can arise from a data driven paradigm. For example; drop down menu selections for our experiment editor are specified in the Part and Lists Manager, the Shot Setup Reports that lists the configurations for diagnostics are specified in the database, the review tool Approval Manager has many aspects of it's workflows configured through metadata that can be changed without a software deployment, and the Target Diagnostic Template Manager is used to provide predefined entry of hundreds setup parameters. The trade-offs, benefits and issues of adapting and implementing this data driven philosophy will be presented.

BACKGROUND AND SYSTEM COMPONENTS

The suite of applications discussed here are used to set up and approve experiments on the NIF. In the context of this paper, an "experiment" is an XML document that stores all of the settings that experimenters are able to configure for an individual laser shot event on the NIF. These settings are functionally associated into "data groups" that define the granularity at which review and approval occurs for the experiment. For example, all of the laser energy and timing settings form a data group, the beam pointing settings are a data group, the target setup is another, and each target chamber diagnostic device setup is its own data group. A "campaign" in the CMT suite is a collection of experiments associated under a given campaign name.

The applications in the suite divide workflow into three broad, nominally consecutive, phases: setup, approval, and readiness. Within the overall lifecycle of an experiment from conception through post-shot analysis, these phases occur in the interval from several weeks to several hours before a shot is taken.

Tools in the CMT Suite

The Campaign Management Tool, CMT, is the experiment setup editor. A Java Swing application, CMT provides a spreadsheet-like interface to the experiments in a single campaign, with each experiment represented in a single column. As data group setups in an experiment are completed in CMT, they are submitted for review and approval. This process is managed by the Approval Manager ("AppMan"), a Java web app that sequences the approval workflow, provides approval status information, and provides links to reports needed for review and approval. Experiment readiness is the evaluation of the state of the NIF facility with respect to the requested configuration for an experiment. Readiness is monitored via another web app, ConfigChecker, which depends on applications outside of the CMT suite (LoCoS and Glovia) that provide up-to-date facility configuration information.

Other applications provide specialized functionality to facilitate key aspects of the suite workflow. The Parts and Lists Manager (PLM) is a database front end through which the project manages most of the setup data option values exposed in CMT selection menus. A close cousin of PLM, the Target Selection Manager (TSM), manages the particular subset of setup menu data having to do with target system configurations. ShotSetupReports is the report generator for the suite, called from within AppMan to access experiment XML and expose setup selections via electronic reports. The Target Diagnostic Template Manager (TDTM) simplifies and shortens the experiment setup process by permitting CMT users to populate reusable setup templates for most of the target diagnostics in use at the NIF. The Pulse Shape Editor (PSE) provides a similar reuse capability for laser pulse shapes, as does the Beam Pointing Assistant (BPA) for pointing setups.

Motivation for a Data-driven Architecture

CMT is the oldest and largest application in the suite, developed around a core architecture that was laid down a decade ago. That architecture has been robust and extensible enough to accommodate tremendous growth in both the number of experiments configured as well as in the number of target diagnostics deployed at the NIF. Nevertheless, over the course of its evolution, both logic and data that were initially defined in the CMT code base have been migrated into other applications and data sources, respectively. The value in moving *logic* into other applications is that it keeps CMT focused as much as possible on being an experiment editor, which pays off in a relative reduction in complexity and the manifold benefits which accrue from that. Thus were born each of

the CMT satellite applications PLM, TSM, PSE, BPA, and AppMan. The benefit associated with moving *data* out of the CMT code base and into external sources is not in simplifying CMT, since in fact this generally adds code and complexity to CMT, but in making the data accessible via well-defined interfaces where it can be updated without the expense of updating CMT. The expense of updating CMT is incurred in the resources required to develop, test, and deploy each CMT release, whereas modifying data stored in a database requires only well-defined update operations and comparatively simple verification testing. Subsequent sections of this paper will examine the benefits and tradeoffs of key thrusts of our data-driven evolution.

PARTS AND LISTS MANAGER

Over time, there are two major drivers for change we experience in the Campaign Management software project: the development of new target chamber diagnostic systems, and the steady stream of resource changes within individual diagnostic systems. New diagnostics generally necessitate matching creation of new elements of supporting software. However, resource changes within individual diagnostic systems require configuration changes rather than design changes (for example, new detector filters to accommodate more energetic implosions). Changes of this sort are readily realized in data-only updates so long as the software is architected to enable this. PLM and the interfaces it provides represent a key realization of our efforts to implement such an architecture.

The Data that Drives the Setup

Setup selections in CMT generally fall into one of two types: numbers within a range (Fig. 1), in which the desired value is typed directly into the field, and selections from a list of discrete entries in a pulldown menu (Fig. 2). Both types are managed internally as lists. For the former, the list typically contains three entries: a minimum, a maximum, and a step size. For the latter, the list contains the set of discrete selections. Note that each list may have a set of additional attributes on each of its entries particular to that list, depending on required functionality.

Figure 1: Numeric setup data entry

Figure 2: Discrete setup data entry

These lists are maintained in relational database tables. PLM provides a web interface through which the list data are interactively managed, and it provides a web service interface through which CMT queries the lists during its startup processing. Once CMT has completed startup it has a copy of each list in memory and it no longer requires a connection to PLM. This approach supports a design goal of CMT which is to permit users to launch CMT then disconnect from the network and continue editing experiments in a campaign. Conversely, it also means that users must restart CMT to see changes in setup data that were submitted after their currently running instance of CMT was launched.

Challenges

The overwhelming benefit of this data-driven approach to managing experiment setup data selections is the low cost of adding or changing data compared to the cost of deploying new code releases to accomplish the same end. Notwithstanding that advantage, the data-driven approach does carry its own challenges, even if they are often of the "nice to have" variety in the sense that they're exposed by the increased ease and rate of making changes.

One such challenge is the sheer proliferation of lists. Having developed an architecture and applications to manage setup data, all new development makes use of PLM. In reality, even though data changes for one list or another happen every week, the vast majority of lists rarely if ever change. List processing is relatively expensive, absorbing on the order of 30 seconds of startup processing each time CMT is launched, so there is considerable overhead for a benefit that is realized in practice on relatively few lists. We recently completed a major refactoring of the list management code and interfaces to achieve better than a 50% reduction in startup list processing time, and additional changes are under consideration.

Another challenge is stale experiment data. Once a user selects a value for a given setting, the experiment keeps its own copy of that value. If a subsequent change in PLM eliminates that value or changes an associated attribute, all experiments that retain copies of the old value become stale, which can cause validation errors. These in turn interrupt approval workflow and necessitate extra effort to update settings and conduct additional setup reviews. Of course such errors could occur as well before PLM existed, but the increased effort required to change available settings under the earlier design coupled

with the existence of considerably fewer diagnostics combined to suppress, relatively, the rate of change and incurred a smaller incidence of stale data.

As it is with software bugs, so too is it the case that data bugs (i.e. stale data) have less impact the sooner we catch them in the experiment lifecycle. Since PLM is the point of entry for editing setup selections, we know directly when the potential for stale data is created by an editing operation that changes an existing entry. We have created a capability within PLM to track experiment references to the values maintained in PLM. Whenever a value is changed, we know immediately what experiments are affected. A job runs every five minutes to process experiments that have been updated in that interval and detect any that have stale references. In a typical day, this processing consumes about five minutes of wall clock time. Currently we expose the data in a report available through AppMan. In coming months we are planning to introduce a new notification service that will keep track of all CMT desktop clients that are on the network. One way we plan to take advantage of this service is to enable notifications to users editing experiments that have stale data which will enable them to correct stale data at the earliest possible moment.

Finally, we experience a minor challenge with PLM data arising from our use of multiple independent computing environments, each with separate deployments of a more or less complete NIF software ecosystem. There are four primary environments: Development, Integration, Formal Test, and Production. To the degree that data was embedded in the CMT distribution, the act of deploying a particular revision to multiple environments imposed a consistent body of setup data in those environments. With our present architecture, users are free to modify setup data as they need to in support of their operating goals in each environment (which do vary between environments). Still, it is generally helpful, and not infrequently necessary, to migrate data between environments. A straight table copy will not suffice, since the primary keys are not portable and the schemas may not match. As a solution for this, we have developed a capability to transfer data through specially formatted Excel spreadsheets that are both written and read by PLM. This process is flexible enough to permit updates to individual columns of specific records or to migrate entire tables.

SHOT SETUP REPORTS

Our report generator, ShotSetupReports, provides reports in HTML and Excel formats that unpack the setup data defined in the experiment XML. There are specific reports for each diagnostic, the laser setup, beam pointing, etc., i.e., each of the data groups, in addition to some other specialized reports.

ShotSetupReports employs a novel and highly flexible data-driven design in which the stored data are scripts that are run against CMT experiment XML and render formatted data to a web page or Excel. In use for approximately two years, ShotSetupReports has by now

matured fully so that actual code deployments are rare and changes to support our ever-evolving target diagnostic systems consist of nothing more than database updates.

There are three principal components to ShotSetupReports: the ShotSetupReports Java application which, when invoked through a URL retrieves experiment XML and executes the identified setup report script; an admin web page that consists most prominently of a setup report script editor; and the report scripts themselves, stored in a database. A fourth component is a web page front end to the application through which any of the available reports can be invoked. This interface is not used as a production tool, since typical users invoke the reports via links in Approval Manager after they've loaded a particular experiment, but the web page provides a simple interface to run reports for any experiment and also provides a place to demonstrate reports under development before they are finalized for production use.

There is not much to discuss in terms of issues or tradeoffs for this application; it is an unqualified success. The scripts are written in HTML and Velocity Template Language [1], providing a rich, flexible, and extensible functional foundation to draw from. Scripts may reference other scripts so very large reports are easily created by aggregating references to existing scripts. Performance, while not breathtaking, is adequate. Most setup reports take several seconds to render to a web page, and some of the very large aggregations, which draw on data sources beyond just the experiment XML, can take 20 or more seconds to complete.

With the codebase for ShotSetupReports having reached a relatively stable maturity, the single aspect that continues to change is in fact the "data" that drives the system – report scripts. The lifecycle maintenance costs for ShotSetupReports are miniscule. Corrections to reports typically consist of updating an xpath used to pull a field from the experiment XML – an operation that usually takes less than five minutes, can be performed without any system down time, and is available immediately to all users as soon as the updated script is committed to the database. With thousands of setup parameters stored in an XML schema that evolves continually in small increments, and exposed by dozens of setup reports, occasional incorrect references are inevitable. For our project to be able to deploy corrections very quickly, usually within minutes of receiving an indication of an error, is a tremendous win both for our software development project due to the minimal resources involved and for our users, whose workflow endures only a brief interruption.

APPROVAL MANAGER

In the experiment lifecycle, Approval Manager drives operational workflow subsequent to experiment setup, facilitating review and approval and finally invoking export of the experiment to the NIF laser control system, ICCS, at which point the experiment lifecycle has moved beyond the campaign management phase and into the

final phase culminating in the laser shot. The transition from setup to review and approval is not a single event but occurs incrementally as each data group is completed in CMT and submitted for review. Depending on the outcome of the review, a data group may be sent back to CMT for re-work and re-submittal.

Since AppMan's introduction in early 2011, the approval cycle it manages has been revised, expanded, and diversified to accommodate the NIF community's increasingly refined approval workflow requirements. Capabilities conceived of and implemented since the initial AppMan introduction include:

- Multi-stage approval workflows with optional or mandatory intermediate approval stages;
- "Virtual" data groups keyed dynamically by prescribed logical conditions in the experiment setup that support both more abstract and more specialized review needs than those dictated directly by data group submittal from CMT;
- Workflow "contingencies" created dynamically by prescribed workflow events that act as a barrier to workflow progress until their driving conditions are resolved;
- Configurable dependencies between data groups so that setup changes in a previously approved data group can force re-review of other, dependent data groups;
- Automated exports of target system data immediately upon approval to enable production of layered cryogenic targets days in advance of the planned shot time
- Configurable message broadcast for one-way communication to logged-in users
- A rich array of admin-level features to facilitate testing, debugging, and approval workflow analysis

AppMan has been designed from the outset to be data-driven and keep important metadata and configuration settings interactively accessible. However, there are some important changes implemented to make AppMan more data-driven whose need was only revealed by experience. For instance, the addition of new diagnostics in CMT impose one set of change requirements on AppMan, but revisions to existing diagnostics may require only a subset of those changes, or no changes at all in AppMan. Being able to characterize these patterns well enough to define a unified collection of metadata that will support either sort of change *and obviate code changes demanding the expense of a new release* required the experience of many change cycles on top of the maturation of our approval workflows.

Most of the above features are realized in key ways via metadata definitions, and virtually all of AppMan's metadata and configuration settings can be accessed interactively through an admin interface. Beyond the imperative to free the project as much as possible from code deployments by designing logic to be driven by easily modified data, the project also benefits, sometimes dramatically, when the unexpected occurs and interrupts operational workflow. Whether it be a bug in the code or

an unexpected external condition that breaks logic, the ability to undo the effects of broken logic by manipulating system data (after thorough analysis of the problem and impacts of the proposed solution!) is extremely powerful and can save valuable time during when it is most critical.

TARGET DIAGNOSTIC TEMPLATE MANAGER

TDTM provides yet another take on the notion of "data driven." TDTM provides setup templates for most target diagnostics that can be configured in CMT. Users populate the fields in the template (drawing on the same setup option data in PLM that CMT uses) then save the completed template under a unique name. The supported diagnostics in CMT each have interfaces to let users load templates into their diagnostic setups, completing an entire diagnostic setup in one simple step. Furthermore, the interface between CMT and TDTM also permits saving a setup into TDTM as a new template for the diagnostic.

TDTM presents some unique challenges owing to the stringent validation requirements CMT applies to users' setups. Since CMT's validation logic is only available to CMT, TDTM cannot apply all of the same error checking and constraints internally to setups that CMT can, opening the door for the occasional error to be propagated into multiple experiments by a single template. The preferred solution to this would be to expose CMT's validation code as a service that could then be called from TDTM, but the low incidence of actual problems from the current architecture does not justify the effort at this point.

SUMMARY

Campaign Management encompasses a diverse array of applications critical to the experiment lifecycle at the NIF. We have adopted a practical policy of data-driven design, both in the end user functional spaces and in application internals. This approach minimizes our need to make changes that require code work and deployments, activities that are inherently more resource intensive and costly than the database updates. Furthermore, when a change is required, it can be affected quickly, minimizing the duration of the interruption to operational workflow.

REFERENCES

[1] "The Apache Velocity Project," http://velocity.apache.org.