FLEXIBLE DATA DRIVEN EXPERIMENTAL DATA ANALYSIS AT THE NATIONAL IGNITION FACILITY*

A. Casey, R. Bettenhausen, E. Bond, R. Fallejo, M. Hutton,
J. Liebman, A. Marsh, T. Pannell, S. Reisdorf, A. Warrick, LLNL, CA 94550, USA

Abstract

After each target shot at the National Ignition Facility (NIF), scientists require data analysis within 30 minutes from ~50 diagnostic instrument systems. To meet this goal, NIF engineers created the Shot Data Analysis (SDA) Engine that uses the Oracle Business Process Execution Language (BPEL) platform to configure analyses and archive results. While this provided for a very powerful and flexible analysis product, it still required software developers to create each unique analysis configuration executed by the SDA engine. As more and more diagnostics were developed and the demand for analysis increased, the development team was not able to keep pace with the rate of change. To solve this problem, the Data Systems team took the approach of creating a data-driven framework that allows users to specify the analysis configuration (analysis routine, inputs and outputs), input data sources, and results archive destinations as data that is stored in the database. The creation of this Data Driven Engine (DDE) has decreased the manpower required to integrate new analysis and has simplified maintenance of existing configurations. The architecture and functionality of the Data Driven Engine will be presented along with examples.

SHOT DATA ANALYSIS ENGINE

The Shot Data Analysis Engine was first deployed on NIF [1,2] in 2008. This highly flexible and scalable analysis framework (Figure 1) features a parallel architecture that:

- automatically triggers analysis when data arrives;
- sequences the analysis workflow;
- provisions data from various data sources;
- maps data to analysis functions written in Interactive Data Language (IDL®) [3]; and
- archives analysis results with their "pedigree" (a record of the data inputs and analysis software).

The Shot Data Analysis Engine distributed architecture divides functionality among the following components:

- Analysis Director - sequences the analysis for each diagnostic;
- Data Mapper - maps data from data sources (Archive, Calibration, NIF Configuration) to analysis, and maps analysis results to the Archive;
- Analysis Server Cluster - executes the analysis routines.

The original Engine's scalable, parallel architecture was accomplished through the use of two key technologies: (1) message queues with Java messaging that dynamically schedule and balance analysis tasks across all available resources — i.e., processes and processors — and (2) a commercial, industry-standard workflow processor called Business Process Execution Language (BPEL) [4] that underlies the Data Mapper component and is used to orchestrate the analysis and perform a data mapping function that integrates external data repositories through Web Services. While distribution through message queues proved robust and reliable and remains in the architecture, the BPEL-based Data Mapper has been replaced with the Data Driven Engine (DDE) to gain more efficiency, robustness, and maintainability.

OPERATIONAL EXPERIENCE WITH BPEL

The BPEL product was chosen because it provided a number of out of the box features that made it very attractive as the orchestrator of the analysis process. The first benefit was the relationship between the existing Archive and the BPEL product. As the Archive is designed around lower levels of the Web Services stack - WSDL, SOAP, and WS-Addressing – BPEL naturally fit into this architecture. We also knew that our analysis capability and requirements would grow significantly as more and more diagnostics were deployed in the NIF target chamber. BPEL was a good match here as the business logic is expressed in XML which allows for easy code maintenance in that it is human readable and there is opportunity to leverage code reuse.

*This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. #LLNL-ABS-631632, # LLNL-CONF-644237

Copyright © 2014 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-139-7

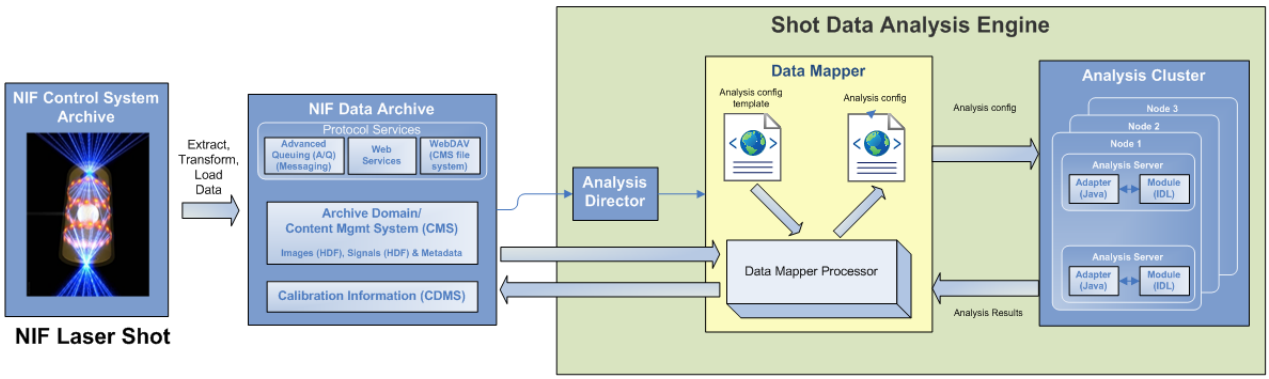

Figure 1: NIF shot-data analysis system block diagram. The data mapping function is highlighted in yellow.

However, as more and more analyses – and thus BPEL flows - were introduced into the analysis engine, our primary use pattern began to change. Analysis was inducing a burst behaviour that was triggering a brief but significant load on the BPEL system itself. The net effect was that occasionally BPEL processes would be starved of resources and workflows would hang, requiring manual intervention in order for them to complete. As the analysis engine tends to run in the early hours of the morning, this can be problematic! While the Data Systems team was able to tune installation and configuration parameters to mitigate some of these issues, specialist knowledge of a BPEL administrator was necessary to implement these changes and as our use changed, so did the configuration set up.

```
-<copy>
<from part="payload"
query="/client:CameraWarpCorrProcessRequest/client:
request/client:iccsTaxon" variable="inputVariable"/>
<to variable="iccsTaxon"/>
</copy>
-<copy>
<from part="payload"
query="/client:CameraWarpCorrProcessRequest/client:
request/client:iccsTaxon" variable="inputVariable"/>
<to variable="taxon"/>
</copy>
-<copy>
<from part="payload"
query="/client:CameraWarpCorrProcessRequest/client:
request/client:dataTaxon" variable="inputVariable"/>
<to variable="dataTaxon"/>
</copy>
```

Figure 2: A simple sample XML specification for copying request data into local variables.

As the number of diagnostics increased and more importantly, the rate of their delivery to the NIF target chamber increased, we found that that we could not reuse BPEL XML flows as efficiently as we had first hoped. Instead of a traditional code reuse model, it was the processes themselves that were being reused. This meant that we were very reliant on dedicated BPEL developers to produce duplicated code. As the initial BPEL flows were relatively simple, creating the BPEL could be done in a time efficient manner. However, as the complexity grew, so did the development time. Eventually, development of the BPEL processes became the schedule driver in the analysis development lifecycle.

THE DATA DRIVEN ENGINE

It was at this point in 2011, that the Data Systems team decided to use the operational experience with BPEL to reassess the initial assumptions when BPEL was made a part of the architecture. There were two key findings from this analysis. The first was that the prime function BPEL performed in the Shot Data Analysis Engine was a data mapping function. It was reading an analysis configuration template and through the use of web service calls, mapping data from the archive to the template and passing the resulting analysis configuration file to the analysis nodes. The second finding was that the Data Systems team needed a solution that did not require a dedicated team of coders to support it. The data analysts themselves had to be able to specify their own data needs. It was from these two findings that the Data Driven Engine (DDE) was created.

The DDE is a Java process that functions much like BPEL. Upon receiving an analysis request, both create the analysis configuration file and pass it to the analysis nodes to perform the actual analysis. Both receive results back from the analysis which they write to the archive. The difference is that BPEL reads and executes logic from a complex XML file that has to include its own error handling, while the DDE executes a reusable Java framework that incorporates mapping and error handling

Copyright © 2014 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-139-7

FROM_DATASOURCE	FROM_PARAM	TO_PATH
setup_parameters	TARGET_VIEW_ANGLE	dante_setup.target_viewangle
setup_parameters	TARGET_LEH_DIAMETER	dante_setup.LEH_diameter
setup_parameters	CHAN_COMP_APPLY_ALIGN	chan_comp.apply_align
fixed_params	TCC_DISTANCE	dante_setup.TCC_distance
fixed_params	CHAN_COMP_FALLPEAK_RATIO	chan_comp.fall_peak_ratio
fixed_params	CHAN_COMP_DWNSAMP_TSTEP	chan_comp.downsamp_time_step

Figure 3: A simple sample DDE specification for copying request data into local variables.

functionality and only requires data from a simple database table. This approach is easier for a human to read than XML and is a lot more logical to debug and troubleshoot. Figures 2 and 3 are excerpts from a sample XML specification and a DDE data table.

While the BPEL in Figure 2 is simple and readable, it involves considerable duplication and overhead to complete the specification and adhere to the XML syntax. Contrast this with the DDE specification in Figure 3. The table is a simple source from / assign to relationship. This format meets both of the key findings. The mapping function is performed simply and a lot more clearly to the coder and secondly, as there is no complex XML to write, the analysts are able to create their own data maps that easily integrate with analysis routines in the Shot Data Analysis Engine.

Another advantage is that the DDE does not require the overhead of processes to achieve code reuse. This, coupled with the DDE self-regulating threading model, controls the servicing of analysis and assures that required resources are available. As a result, the DDE moderates analysis bursts and does not itself induce burst behaviour as occurs in the BPEL system.

It must be noted that there are drawbacks with this system. The DDE data specifications are not a programming language. The lack of higher level programming constructs such as loops, conditional statements etc., can result in complex data specifications that are not easy to follow months after the original writing. This significantly affects maintainability. Another drawback to consider is the cost of developing the new workflow engine.

THE NEW DDE AND GROOVY

The maintainability issue of the DDE called into question its long term feasibility as a user friendly solution. The DDE is certainly simple in concept but if the resulting specifications are more complicated than the original XML, the benefit is limited. The assessment of the team was that the DDE needed to be able to specify higher level programming constructs such as functions and that these functions would execute like a macro within the existing Java based DDE and Archive Viewer applications. Thus, the requirements became; the macros would be human readable, be editable by a data analyst using the Archive Viewer, would have a Java like syntax and would not require any compilation or development environment. After evaluating a number of potential scripting extensions, the Data Systems team settled on Groovy.

Groovy [5] is an open-source, general-purpose scripting language that runs on the Java Virtual Machine (JVM) and, for people with some familiarity with Java, has very little learning curve.

The incorporation of Groovy java scripting gives the data analysts the ability to extract complex data specifications into macro code while keeping basic mapping in data. Additionally, DDE developers can use macros to quickly provide needed functionality in real time without requiring a formal code release.

BENEFITS ANALYSIS

As the DDE evolved in technology, the number of specifications needed to define the data mapping for a new analysis was reduced. The original BPEL and DDE implementations required 4 separate specifications (Figure 4); Analysis Configuration Template, Archive Object Definition, Process Logic, and Interface Specification. While the Process Logic accounted for most of the effort in terms of schedule, the other 3 elements also contained duplicate information that presented an opportunity for further simplification.

The latest DDE incarnation requires an interface definition (where the data is coming from and going to), the definition of the archive object class, and if necessary, the process logic of higher level functions.

For an analysis of moderate complexity, based on our current prototype, this is expected to reduce the time to integrate a new diagnostic analysis from 15 weeks to about 6 weeks (Figure 4).

ISBN 978-3-95450-139-7

Copyright © 2014 CC-BY-3.0 and by the respective authors

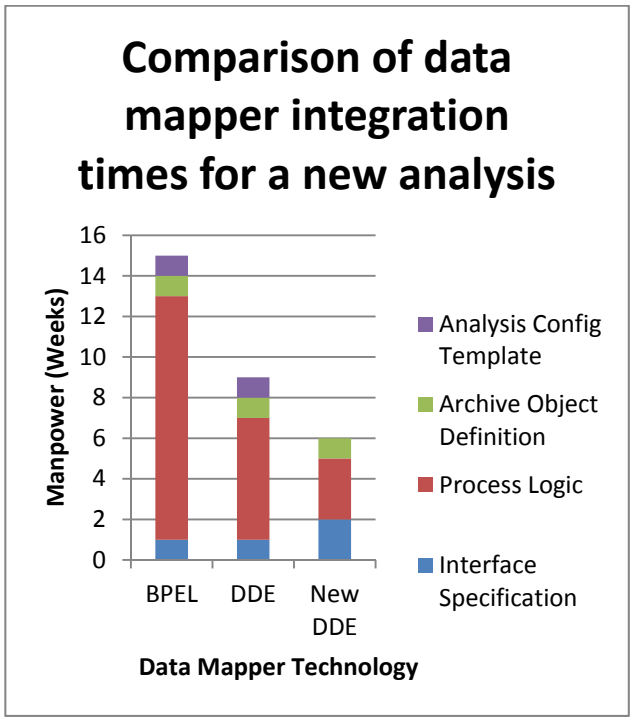

Figure 2: Each iteration of the Data Analysis Engine has reduced the manpower required to integrate a new analysis.

SUMMARY

In the field of SW development, the general strategy is to use COTS products wherever possible in order to minimize local development effort and to maximize the capabilities and experience of another development team.

However, there are times when the replacement of COTS products with custom software yields significant benefits in terms of tailored functionality that fully meets the user needs and makes better use of development dollars.

In migrating from BPEL to the DDE, the Analysis team at NIF achieved:

1. more efficient re-use of existing capabilities and functions;
2. simpler, user-specified data mapper configurations;
3. increased transparency and maintainability of data mapper configurations;
4. load balancing that handles peak loads predictably and reliably;
5. less manpower to add a new analysis to the Shot Data Analysis Engine;
6. fewer dedicated, specialized software developers.

With the new DDE, the team is expecting to be able to achieve:

1. greater simplification and maintainability of data mapper configurations;
2. additional decrease in manpower needed to add a new analysis to the Shot Data Analysis Engine.

REFERENCES

[1] E. Moses, et al., "The National Ignition Facility: Path to Ignition in the Laboratory," Fourth International Conference on Fusion Sciences and Applications," Biarritz, France, September 2005.
[2] The National Ignition Facility Web site, https://lasers.llnl.gov
[3] Gumley, L. E., Practical IDL Programming, Morgan Kaufmann, 2001.
[4] Oracle BPEL Process Manager web site, http://www.oracle.com/technetwork/middleware/bpel/overview/index.html
[5] Groovy web site, http://groovy.codehaus.org/

Copyright © 2014 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-139-7