Avoiding the "Inner-Platform Effect" by using Protégé

Ron Schultz – Metavante Image Solutions

Today there exists little in the way of software tools to effectively manage the configuration of software systems having the following attributes:

- The software system is large and complex, involving many hundreds of thousands of lines of code and hundreds of modules and components.
- The system is intended to be very configurable to the varying demands of multiple customers or to the constantly changing demands of a single organization, or both.
- The system has many dependencies with external systems outside of the control of the software application. Such dependencies may include, for example, interactions with databases, web services, or other complex software systems.

Attempts to address such configuration issues often yield a syndrome referred to as the "Inner Platform Effect." Quoting the URL http://worsethanfailure.com/Articles/The_Inner-Platform_Effect.aspx :

> *"The Inner-Platform Effect is a result of designing a system to be so customizable that it ends becoming a poor replica of the platform it was designed with. This "customization" of this dynamic inner-platform becomes so complicated that only a programmer (and not the end user) is able to modify it."*

The quoted URL also provides an example of a possible outcome of a system exhibiting the syndrome.

> *"The loan-origination system that Mario was dragged into was designed to be revolutionary: instead of relying on a programmer to make changes to the database, a user would simply need to make a few changes via the 'Data Structure Modeler' (DSM).*
>
> *As you may have gathered, the DSM organized data using a structure it called a 'Table'. These tables had one or more Fields, with each Field having a specific 'Data Type'. A single datum was then stored as a 'Field Value.' All of these were stored inside of a relational database.*
>
> *Ironically, Mario was brought into the system because the client wanted to add a new Field to a form."*

This presentation discusses the lessons learned in using Protégé to develop a "Configuration Editor" for a complex software system that avoids the "Inner-Platform Effect."

Background

In October, 2003 the U.S. Congress passed the Check Clearing Act for the 21st Century. The Act became law in October, 2004. The Act allowed a financial institution (e.g., bank) to print a document that included the image of a check, and present that document (referred to as an "Image Replacement Document") with the same legal weight as presenting the original check.

The above is an example of an Image Replacement Document. The document is also commonly referred to as an IRD, and a Substitute Check.

The Act's effects on the U.S. banking industry have been dramatic. In 2000, nearly 60 billion checks were exchanged between financial institutions. This exchange required physically transporting checks between banks. Transport was conducted using vehicles such as armored cars, courier services, Federal Express, and corporate jets. In less than 3 years, millions of checks a day are now moved electronically. Over 50% of the dollars in the U.S. check payments system are now moved electronically. The Act created a large demand

for software to support the transition from moving physical paper (checks) to moving electronic images and data across high-speed data networks. The check processing systems that processed physical paper checks also had to be reengineered. These systems have evolved over the past 40 years and tend to be heavily customized within each bank.

U.S. financial institutions are also confronted with the fact that people are writing fewer and fewer checks every year. In fact, check writing in the U.S. is declining at the rate of 5-7% per year, and the rate is accelerating. No bank wants to spend millions on internally developing new software to satisfy the needs of a declining market. In order to mitigate these costs most banks have sought commercial, off-the-shelf, software solutions to meet their needs.

Metavante Image Solutions is a provider of such solutions. Over the past three years we developed a solution that addresses the complex workflow and custom processing required to provide check image exchange to financial institutions. The check exchange system is installed in 20 of the largest banks in the U.S. The exchange system started processing a few hundred check image exchange transactions a day starting in February of 2004, and today (April, 2007) processes over 20 million transactions a day. But we, and our customers, have found configuring the exchange system is difficult and error-prone.

The exchange system today contains over 1200 settings. The system's **Configuration Guide** is almost 200 pages in length. These configuration settings aggregate into numerous functional, logical, and temporal groupings. Many settings drive or require coordination with configuration settings in external systems, such as database management systems and check processing systems. Adding new settings, deprecating old settings, and verifying the integrity of each setting change into a configuration is often a major undertaking. Testing these settings can take months and involve man-months of effort.

Beyond configuring the exchange system for known and understood activities, a number of customers have found ways to configure the exchange system to perform activities never originally contemplated in the original design. These configurations have proved extremely valuable to these customers (netting them significant new revenue) but often these settings created unanticipated side effects requiring unplanned software modifications.

Why Protégé?

Over the course of my career I have managed, developed, and prototyped numerous efforts to build system configuration editors. The requirements for any such configuration editor should include, as a starting point:

- Modeling dependencies
- Easily developing export routines to generate configuration files in a manner suitable to support the configured system. This may include, for example, SQL database scripts, XML files, Java property files, and Windows registry entries
- Providing documentation, such as HTML or RTF files
- Managing and tracking configuration changes
- Taking configuration snapshots
- Performing version compares
- Search capabilities
- Validation capabilities
- Providing role-based access and controls

Each editor has had its unique strengths and weaknesses, but none proved to meet all the desired requirements. Each editor's design failed from some fundamental architectural flaw. These architectural flaws included:

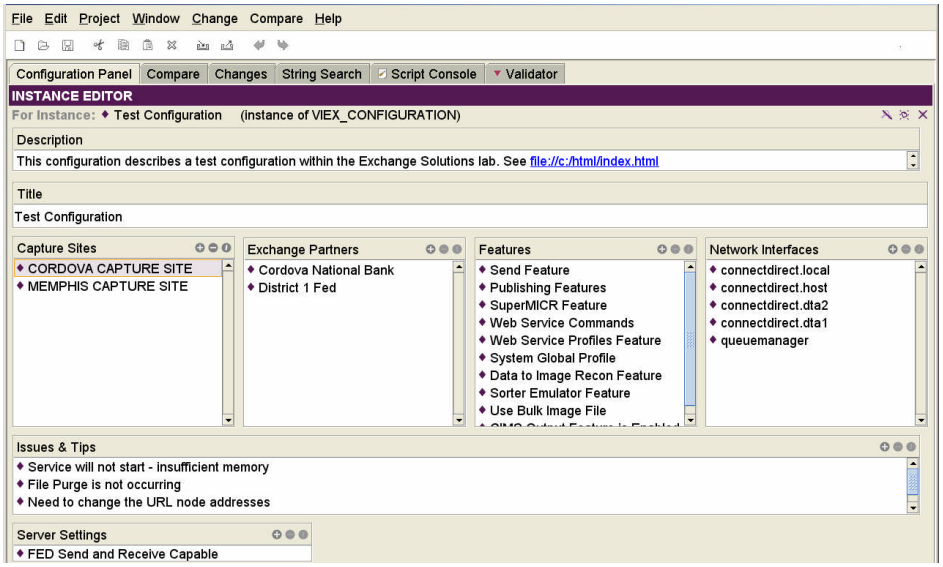
- Poor performance (e.g., poor performing queries)
- Inability to provide role based access control
- Lack of maintainability and modifiability (e.g., could not address changing user requirements quickly enough)
- Lack of portability (e.g., reliance on some proprietary (usually database) technology)

I started exploring Protégé in 2003. At the time few on my team understood the synergies in pursuing a system configuration using the mindset of ontology creation. That lack of knowledge has since been

addressed by a growing awareness of the value and purpose of ontologies. Articles and conferences on Web 2.0 and The Semantic Web are promoting an awareness and understanding of ontologies to all industries, including banking.

I view using Protégé as a Configuration Editor as an “off-label” use of Protégé. Where “Off-label prescribing” is the physician practice of prescribing a drug or device for a purpose different from which the product is approved, I suggest a similar off-label prescribing of Protégé for the configuring and validating of complex software systems.

Modifying Protégé for use as a configuration editor was straight-forward. First, building the Protégé ontology from the settings in the check exchange system resembled an archeological expedition. Each setting carried some history, and often undocumented, interaction, with the rest of the exchange system. Surfacing and documenting this information into the ontology provided for a deeper understanding of the exchange system than was previously available. As new relationships were discovered, the model could be modified and presented quickly to users for feedback, without modifying their exchange systems running “in production.” We started to discover “features” within the ontology that we never knew were in the exchange system itself (but that some of our customers had inadvertently discovered and exploited). As the ontology grew in depth, we were able to start to build test cases for the exchange system through queries into the ontology. We also discovered that the ontology itself could be used to provide troubleshooting and debugging information to better assist our customers in problem resolution. Addressing the requirements mentioned above proved simple within Protégé. Many of the requirements were already addressed by available plug-ins (e.g., the PROMPT Tab). As a side benefit, the architecture of Protégé avoided many of the architectural limitations I had encountered in prior efforts.



The above form depicts the Configuration Editor Main Panel. Of note is the ability to configure features, and then validate newly configured features in the overall context of the application using the Validator Tab. The modifications required to Protégé to provide this Configuration Editor are minimal.

Summary

The “Inner-Platform Effect” yields another system (i.e., an inner software platform) that often replicates the same problems the system was intended to solve. Using Protégé avoids this effect by approaching the activity of system configuration editing using technology and processes that differ significantly from conventional software engineering methods. While the process of creating software shares many activities with building ontologies, they differ enough to avoid the “Inner-Platform Effect” when constructing a configuration editor for a complex software system. Our efforts to date using Protégé have confirmed this to our satisfaction.

The Configuration Editor discussed is currently being tested by a number of our customers. Our experience to date has shown Protégé a competent tool to address the configuration issues presented. We still have much to learn, and we look forward to continuing to use Protégé to address the complex, but highly valued, business needs of the financial institutions we support.

About the Author

Ron Schultz is the Director of Development for Exchange Solutions, for Metavante (headquartered in Milwaukee, WI). He and the Exchange Solutions team are based in Memphis, TN. He has over 30 years of experience in software engineering and consulting. He can be reached at ron.schultz@metavante.com.