Experience Report: Business Automation with Distributed Objects

Jason Rogers, Dean Mackie, Angus MacArthur
Ontario Teachers' Pension Plan Board
5650 Yonge St
Toronto, Ontario
Canada M2M 4H5
Tel: (877) 812 7989
Fax: (800) 949 8202
http://www.otpp.com
jrogers@otpp.com
dmackie@otpp.com
amacarth@otpp.com

Abstract

A large financial institution was faced with the challenge of having to process twice the normal yearly workload without increasing their workforce. The challenge was met by reusing components of an existing object oriented application in a distributed, fully automated configuration. That solution is discussed, along with subsequent system architecture evolution, resulting improvements to development practices, and the ripple effects of changing the culture of senior management that it was designed to serve.

Introduction

The Ontario Teachers' Pension Plan, with over $73 billion in assets, is the single largest pension plan in Canada. It is responsible for the pension income of approximately 153,000 elementary and secondary school teachers, and 77,000 retired teachers and their families. The customer service department enjoys a reputation within the pension industry for using leading-edge information technology to provide direct and strategic support to internal core businesses, resulting in exemplary service to plan members

The Benefit Entitlement System for Teachers (BEST) has been in production since December 1995. It is an internally developed Smalltalk application that makes extensive use of object-oriented practices, such as architectural layering and automated regression testing[i].

The focus of BEST is on business process automation and system integration. For example, the system has the ability to automatically detect an incoming phone call from a client, pre-fetch the necessary data and present it to the customer service agent. At the request of the client, the agent

has the ability to generate personalized estimates with several different retirement scenarios. Using our workflow system, an electronic version of an estimate letter is passed to our Quality Assurance department for manual double-checking. Once approved, it is passed to the mailroom where the document is electronically archived, printed, and mailed.

In April of 1998 an early retirement incentive for teachers was announced. We were faced with the prospect of processing twice as many retirements as any previous year and handling many more phone calls and requests for personalized estimates. Failure to deal with the increased demand for retirements would have resulted in serious financial hardship for some of our clients. Our employees require twelve months of specialized training to become proficient, so adding more people wasn't an option.

What follows is a description of how objects were used to solve the problem and how the solution resulted in several unexpected changes. The system architecture changed from client-server to distributed objects; our development practices shifted from automating individual steps in a business process to automating full business processes; and the solution had the ripple effect of changing the culture of our company including its senior management.

The Initial Solution

When the early retirement announcement was made, the development group suggested automating the processing of these retirements as a way of meeting the increased workload. Existing objects would be reused because regression tests that tested the end-to-end processing were running cleanly. However, senior management was concerned because pensions would be paid without human intervention. This was a valid concern because in general people only retire once in their lifetime so getting their pension right the first time is very important. The risks were deemed to be manageable because low risk cases were selected, very few problems had ever been detected interactively, and a system to automate checking pensions was already on the drawing board.

Once authorized, a solution was assembled that used an external workflow system for work distribution, reused domain objects to calculate and process the pensions, and leveraged a custom correspondence framework[ii] to generate a confirmation letter. All the necessary logic was built into a specialized application that ran on several computers.

We made several observations. Using our workflow system to delegate work was very problematic because each program was working with cached workflow data that quickly become stale. This caused contention problems such as one case being simultaneously processed by two programs. Stressing the objects continuously raised problems with garbage collection that were corrected by programmatically activating the garbage collector. The increased database contention caused by many clients updating the same tables also exposed a bug that resulted in duplicate relational keys being assigned. Even with these problems, the project was an overall success because each of the 8,000 teachers who retired that summer was paid on time.

After repeating the process the following summer, it became obvious that full automation of this process should become part of day-to-day business, not just at peak periods. What follows are summaries of how the system architecture, development practices, and the business culture have adapted to the new focus on full automation.

Changes to System Architecture

The system architecture of BEST changed from a fat client-server model to an architecture based on a network of lightweight components distributed across the network. The main force behind the transition was the creation of a framework for task scheduling that used a distributed infrastructure. While migrating towards a distributed architecture several interesting problems arose. To solve these problems, objects needed to be enhanced to become more resilient to the forces of distribution.

The goal is to leverage our existing desktop computing resources with minimal changes to our client application. This has led to our **Dispatcher**/**Executor** framework. The Dispatcher is a broker residing on a remote host that is responsible for retrieving and dispatching work to a pool of Executor enabled clients. The Executor is an extension to the desktop client code giving it the ability to participate in a distributed architecture. It is able to receive jobs from a remote Dispatcher, and delegate them appropriately to our business logic without any user intervention. Upon completion of a job, the Executor notifies its Dispatcher that it is available for more work. Dispatcher/Executor intercommunication was achieved using a vendor-supplied CORBA solution. This mechanism is being reused to web-enable our internal applications.

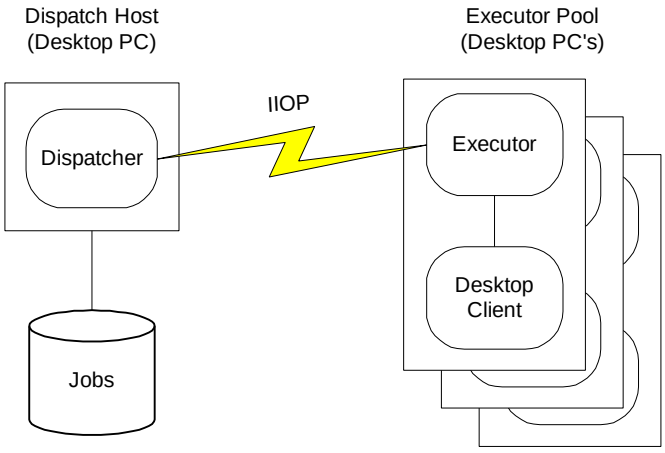

Figure 1: The Dispatcher/Executor framework

The process of deploying systems with distributed objects raised several new issues that previously hadn't been a problem in our client-server model. As more projects automated different tasks, it became clear that several different modes of process activation were required. The activation modes we currently support in production are: event based, scheduled, scavenging, and ad-hoc. Network hardware and software configuration problems had a fatal effect on our system. Even though our physical distribution boundaries consisted of multiple floors within one building, network problems caused inter-object communication failures, primarily due to time-outs. Application-level re-try was considered, but discarded because our strategy was to assume the worst possible case of partial failure and build facilities for easy detection and reporting of problems. It was felt that re-try could compound problems as easily as fix them. Moving from a single-threaded client environment to a multi-threaded environment forced us to protect objects that weren't thread-safe. In particular, our database façade singleton was not thread-safe and had to be protected with a semaphore.

Our objects had to become more robust as they were distributed. Objects had to become self-diagnostic because we discovered that even one runaway server object could sabotage an entire process. For example, objects that couldn't distinguish between a query that returned nothing because there was no database connection and a query that returned nothing because nothing matched the search criteria. To correct the problem logic was added to check the state of critical

external resources. Another aspect of self-diagnosis is that some objects were enhanced to perform heuristic checks previously done via human intervention.

Development Process Changes

Internal development processes have changed to accommodate development of systems with this new architecture. A feedback loop has been developed to make all systems more robust by upgrading core components to handle problems that only arise under extreme conditions, such as high volume or object distribution. Testing procedures have changed to account for the system architecture changes. There is an increased focus on reuse with the help of a framework for modeling business processes as objects.

In an attempt to make the entire system more robust, problems that were only detected when the application was subjected to the new stresses of distribution are fed back into core components. The database façade problem and solution mentioned above is an example of this positive feedback loop. We also uncovered calculation anomalies while processing large amounts of data. Correcting those anomalies has made the system more accurate.

Testing processes have also been affected. Regression testing is now supplemented with statistical sampling to determine confidence levels. Our internal test tool was distributed and now has the ability to run regression test suites 33% faster. Another benefit of distributing the test tool is that issues that only arise upon distribution arise sooner in the development cycle. Our typical development progression involves starting with component level testing and quickly building towards fully integrated testing.

This focus on automated systems with no user intervention has changed the way we determine when a system is ready to deploy. Previously we had the luxury of people manually checking the system results so any systematic problems in a new release could be detected and fixed quickly. Now we have a graduated audit process that still begins at 100% audit but quickly drops down to 10% with most applications. We use statistical sampling of random data sets to determine confidence levels. Our standard goal is 99% confidence that a maximum of 2% of the target population is in error.

The introduction of our distributed framework highlighted the need for a common layer serving up business functionality to both remote objects and our original user interfaces. Ideally, this

layer would be modular and flexible enough to support changing business processes. This led to the introduction of our **Process Framework**[iii]. Simply put, this framework enables us to create Step objects that we can compose into a flowchart-like structure. These steps in turn delegate responsibility of implementation down to our business domain code. By doing this, we can extract business processes embedded within our domain logic, and expose them as an application layer. Having our business processes available in the application layer makes our core domain code is more resilient to changes in business procedures. Steps can be rearranged, inserted or removed without impacting our domain-level components.

One of the more positive effects of the process framework is that the turnaround time for delivering new systems with fully automated functionality has decreased because new processes are assembled from existing components. It has become possible to develop and deploy small applications in two and three week cycles because of the ability to leverage existing process objects.

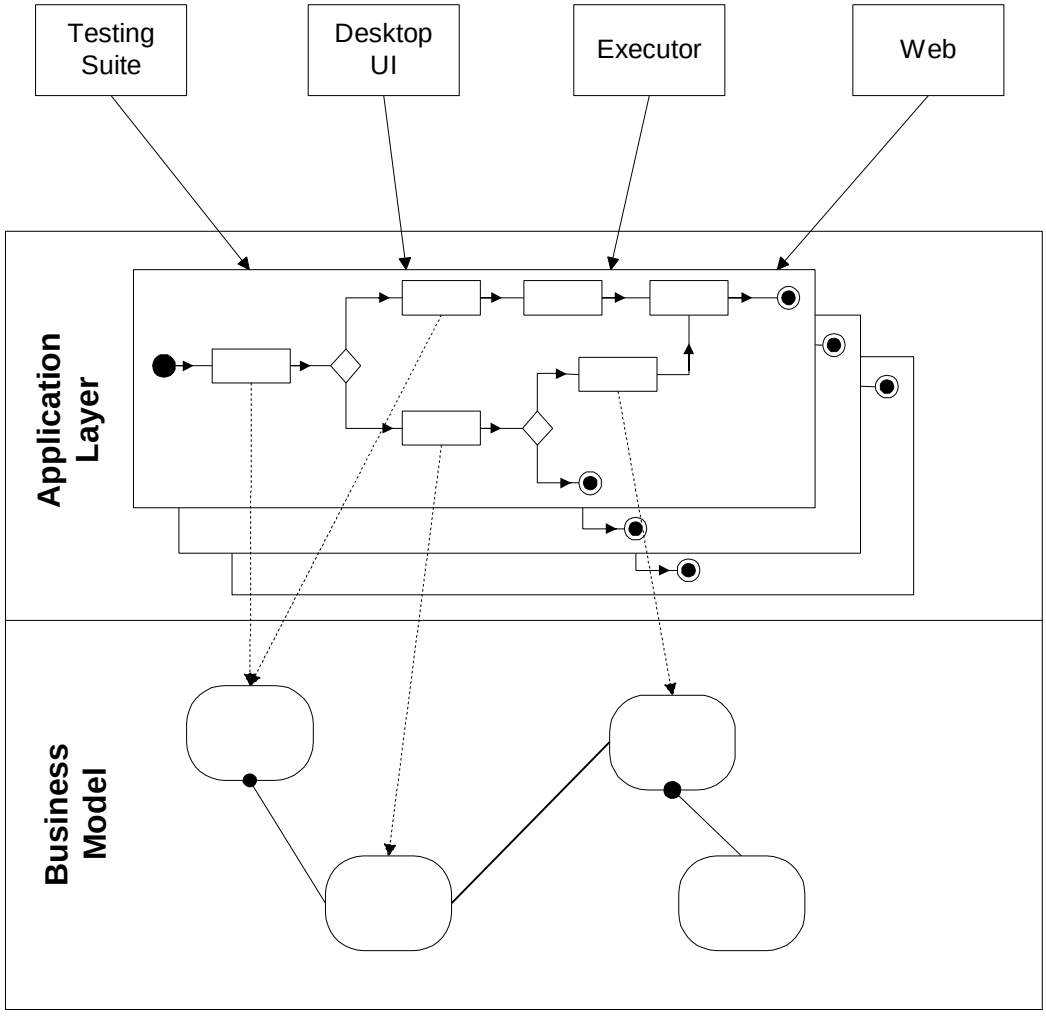

Figure 2: Various clients accessing business functionality through the application layer. Steps within the processes delegate responsibility to our feature-rich business model

Business Process Changes

Senior management has undergone a culture shift as a result of changes to system infrastructure. Fully automated systems have enabled large scale, proactive customer service. Many other manually-intensive tasks have been automated and completely offloaded from employees. The casework procedures were adapted to use more information from automated system.

There have been several projects deployed which provide proactive customer service. One system automatically corrects a pension when more timely information is received. Another generated 30,000 personalized letters detailing the impact of prospective legislative changes.

The ongoing focus on offloading work from customer service agents has become more evident as new projects arise. Problems that previously have been handled by adding interactive functionality are now re-examined for the possibility of automated processing. For example, when forced to deal with processing thousands of election forms for a benefit enhancement, the decision was made to capture the request interactively and have an automated back-end process do the work.

As automated systems become more prevalent, senior management indicated the need to know why an automated process had rejected a particular piece of work. In response to this, reporting was integrated into the workflow system. When a customer service agent is faced with completing some work that the automated system has rejected, the workflow system presents a checklist of things to correct before processing can be completed. This empowers customer service agents to be knowledge workers, and eliminates clerical work.

Conclusion

A business challenge was met by reusing components of an existing object oriented application in a distributed, fully automated configuration. This solution forced a fundamental change to system architecture, improvements to development practices, and changes to the business culture. The evolution of distributed components was critical to making our systems scale. Development

practices became more attuned to reuse and component assembly. Senior management immediately recognized the potential of full process automation to enable the vision of immediate, personalized service.

A new business problem is causing another fundamental change in how systems are developed, deployed and used: customer self-service via the Internet. The lessons learned over the past five years have been of great value as we face the challenges of web-enabling our business.

---

[i] Mackie, Dean, and Xue, Martin. *Automated Testing of an Object Application.* Experience Report for OOPSLA 1998.

[ii] Mackie, Dean, et al. *Applying a Design Pattern: Automatic Correspondence Generation*. Experience Report for OOPSLA 1999.

[iii] Manolescu, Dragos-Anton. *Micro-Workflow: A Workflow Architecture Supporting Compositional Object-Oriented Software Development* PhD thesis
Computer Science Technical Report UIUCDCS-R-2000-2186
University of Illinois at Urbana-Champaign, October 2000, Urbana, Illinois