A Case study in Application Integration

Stijn Van den Enden

Business Integration Company
PB 494
2000 Antwerpen 1, Belgium
stijn@bico.be

Erik Van Hoeymissen
Gregory Neven*
Pierre Verbaeten

DistriNet Research Group
Dept. Computer Science, KU Leuven
Celestijnenlaan 200A
3001 Leuven, Belgium
{erikv, gregory, pv}@cs.kuleuven.ac.be

Abstract

In this position paper, we present an architecture that we used in a case study on Enterprise Application Integration (EAI). The architecture encapsulates all business logic in a workflow, and uses intelligent adapters to provide for the "glue" that links the external applications to the workflow. These adapters are able to transform XML message formats into other data formats or into objects. They are intelligent by which we mean that they are able to take different actions based on the content of the message. To communicate with the applications, the adapters can make use of the services offered by a message broker.

1 Introduction

The case study elaborated in this paper is situated in the context of an enterprise integration project after the merger of two large companies. The overall goal of the project is to enable the sharing of data and business processes among any connected applications and data sources in the "new" enterprise, without having to make sweeping changes to the existing applications or data structures. Moreover, the integration allows the development of new services that are supported by the existing portfolio of backoffice applications with a minimal effort.

2 Problem Statement

The overall problem is presented in figure 1, where we consider the development of a new application that makes use of existing legacy applications of both organizations. We want to provide an architecture that makes it possible to implement the business logic of such an application as a workflow that delegates specific subtasks to the backoffice applications and allows interaction with the users via the frontoffice applications.

In this case study we consider a frontoffice application developed in Sun's 4GL [1] that makes it possible for an employee to log in with a specific role, receives tasks from the workflow system and displays information in a user interface. The backoffice legacy application we consider here is the customer management system implemented in LINC [2]. This system is responsible for the storage and processing of all customer related data of the insurance company.

* Research Assistant of the Fund for Scientific Research – Flanders, Belgium (F.W.O.)

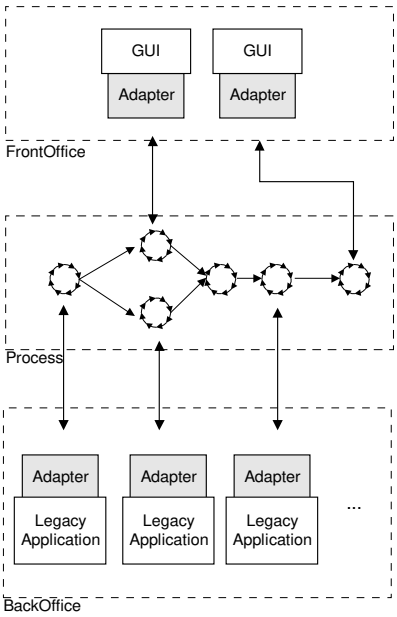

Figure 1 The overal integration architecture

3 Solution

In this section we present the architecture that we used to solve the EAI problem. The architecture encapsulates all business logic in a workflow engine, and uses intelligent adapters to couple back- and frontoffice applications to the workflow. The adapters are able to transform XML message formats into other data formats or into objects and they are able to take different actions based on the content of the message. To communicate with the applications, the adapters can make use of the services offered by a message broker.

3.1 Workflow

The workflow describes the flow of data between applications, and encapsulates the overall business process. The flow focuses solely on the business process by making abstraction from the procedure to interact with each of the backoffice systems. All data flowing between the different activities is formatted in XML. Both the frontoffice and the backoffice understand XML, and interact with the workflow system by exchanging XML messages. The reasons to choose XML as the language for the description of the data flowing through the workflow are:

- interaction with various types of clients including web clients using HTML, and mobile clients using WML;
- standardized transformation capabilities with XSL/XSLT;
- easy integration with legacy applications using of the shelf adapters, and with forthcoming business to business services such as electronic marketplaces and Application Service Providers;
- built in validation capabilities using DTD, XML Schema, etc.;
- XML provides a technology independent representation of the data. XML data can easily be mapped to program structures for processing in any programming language.

The workflow process engine of choice is Sun's Forte Conductor (a part of Forte Fusion [3]). Forte Conductor is a workflow product that essentially consists of three components: a workflow engine, a history and analysis component and a graphical modeling tool. The heart of the Forte Conductor workflow product is formed by a workflow engine that manages the processes. The

engine decides what step a process is in, who should be notified next, who needs to do work next, which application needs to be called at a certain point in time, etc. Client applications register on events with the engine to tell them when there is work for them to do. Each client has a specific role that determines for which activities it receives events. The history and analysis component keeps track of the current state of all processes in order to determine what happens next, to be able to recover in case of a failure and for auditing reasons. Conductor also provides a graphical tool that allows developers to draw the process description as a set of activities connected by lines that represent the flow of control. Control flow design is supported by features like recursion, routing rules, triggers and timers.

In our architecture we use the conductor workflow engine to interact with the backoffice and frontoffice through adapters. The details about these adapters are explained in the next section.

3.3 Adapters

By using intelligent adapters (i-adapters), all details about the interaction of the workflow system with other applications are kept out of the workflow layer. Each separate system that interacts with the workflow engine requires a specific adapter. An adapter logs in on the conductor engine with a dedicated role. The role links the adapter to the process engine and makes sure that the adapter only receives tasks the underlying application can handle. The adapters can be divided in two categories: backoffice and frontoffice adapters.

3.3.1 Backoffice adapter

As shown in figure 2, a backoffice adapter consist of a robotic client, an integration workflow, MQSeries Integrator and the MQSeries messaging service [4].

A backoffice adapter is coupled to the workflow system through a robotic client (RBC). The coupling is based on a contract that defines a complete XML interface for the services offered by the backoffice application. The contract defines the input parameters, the output parameters and possible exceptions. It fulfills a central role in the integration architecture. Without an agreement on the structure and content of the XML messages an unambiguous communication would not be possible.

The robotic client automatically receives tasks from the workflow engine whenever they become available. All information needed to call the specific service in the backoffice is supplied by the workflow engine as an XML message. The robotic client will validate the ingoing and outgoing messages against the contract, so errors are discovered early in the processing. If the message is valid, the robotic client will start a specific activity like for example retrieving customer information from the backoffice application.

The operation that the robotic client wants to execute on the backoffice requires itself a sequence of steps that are modeled as a workflow (the integration workflow). This workflow typically deals with issues like error handling, but it could also be used to extend the functionality of the backoffice application. As before, the integration workflow will delegate operations to robotic clients, but in this case the operations are such that they can be executed directly on the backoffice (i.e. they don't have to be modeled as a workflow). The robotic clients make the XML message ready for processing by MQSeries Integrator (MQSI).

The backoffice adapter sends messages to the customer management system through IBM's message queuing product MQSeries, which provides exactly-once asynchronous message delivery. While MQSeries only supports point-to-point communication, IBM's MQSI extends this functionality by providing a message brokering service with features like intelligent routing, publish/subscribe functions and mapping between different message formats.

In this particular case study, MQSI is used to convert the hierarchically structured XML messages into flatfile messages understandable by the LINC system. Unfortunately, MQSI does

not support XSLT to perform this mapping, so ESQL has to be used. After this mapping, MQSI routes the message to the right queue for the LINC system.

The communication in the opposite direction is completely analogous: the LINC system puts a flatfile message on the MQSeries queue, MQSI translates it into an XML message and routes it to a robotic client that feeds it to the integration workflow.

To make things more concrete, we consider as an example the integration workflow that models an update operation on the customer management system. We will not go into the details, but because of specific linkage of the system to MQSeries, MQSeries cannot be used as a guaranteed message delivery channel. To solve this problem, we model the update operation as the following sequence of activities.

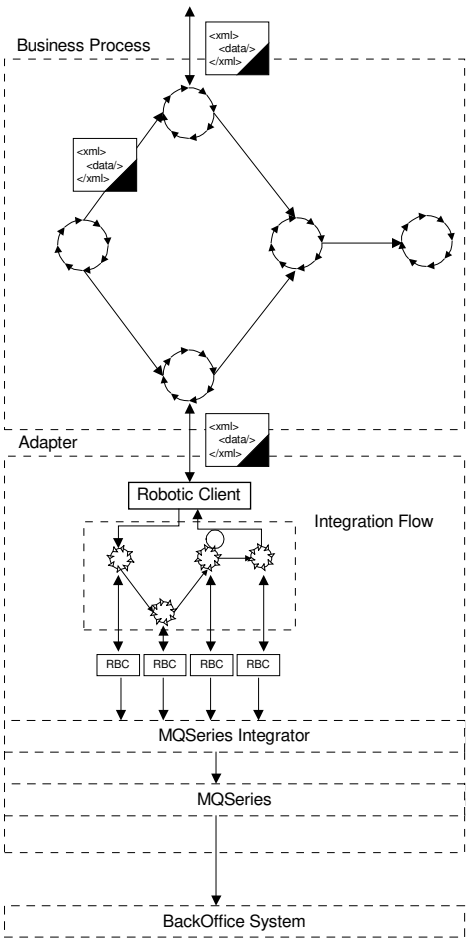

Figure 2 Overview of the backoffice adapter

1. A message is sent to the customer management system through MQSI and MQSeries. A unique ID is sent with the operation request. This ID is logged into a database at the side of the backoffice. The database stores the ID and the return message of each request, in other words the database contains a log of all operations.
2. When the return of the operation is successfully received, the message can be forwarded to the adapter. However, when the adapter doesn't receive the return message on time, the workflow process should check whether the operation has been received by the backoffice and has been completed successfully. This can be accomplished by interrogating the database with the ID of the operation. The possible scenarios are:

- the operation has completed successfully and the return message should be retransmitted to the adapter;
- the operation is not received by the backoffice system and the original request should be retransmitted;
- no response is received from the database on time. In that case the interrogation of the database is retried several times before an error message is posted that propagates back to the user and to a specific system administrator.

3. The operation has completed successfully or is aborted. Now the log ID can be removed form the database in the backoffice.

3.3.2 Frontoffice adapter

In this case study, we consider a frontoffice application implemented in Sun's 4GL. Unfortunately, most OOPLs don't natively support serialization of object data to XML and vice versa. As shown in figure 3, a specific frontoffice adapter takes care of this.

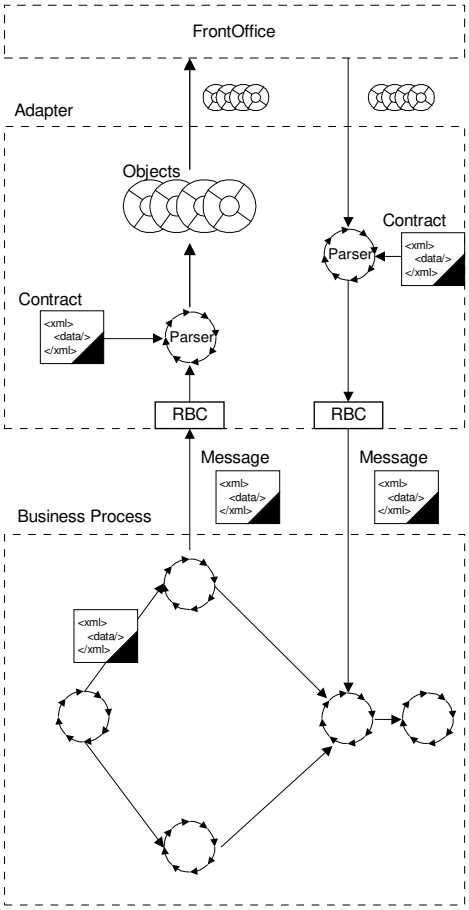

Figure 3 Overview of the frontoffice adapter

The frontoffice adapter is coupled to the workflow system through a robotic client that receives tasks from the workflow engine. The interactions are specified in a contract. The contract is an XML document that defines the structure of the messages and the data types that can be contained in an XML tag. Based on the contract, the robotic client will use a parser to validate an incoming message and deserialize it into objects that can be used by the frontoffice application.

The data type information is essential to make the mapping of the data in the XML message to the object attributes possible. For outgoing messages, the reverse action is taken and the objects are serialized into XML messages.

4 Conclusion

EAI problems are very specific and extremely complex, no matter what some of the software vendors may pretend. In practice, the coupling of a business workflow with intelligent adapters turns out to be a very flexible and solid integration architecture. The encapsulation of a specific integration workflow in the intelligent adapters prevents the business workflow from becoming too complex. The architecture can be extended to integrate various legacy and frontoffice applications.

5 References

[1] http://www.sun.com/forte/4gl/
[2] http://www.unisys.com/marketplace/linc/
[3] http://www.sun.com/forte/fusion/
[4] http://www.ibm.com/software/ts/mqseries/