18

INTEGRATING A WORKFLOW ENGINE AND A MOF REPOSITORY TO AN OPEN SERVICE PLATFORM

Prof. Msc. Cláudio R. M. Silva
UFRN / UNICAMP
crmsilva@dca.fee.unicamp.br

Prof. Msc. José A. Soto
Universidad Tecnologica de Pereira / UNICAMP
josoto@dca.fee.unicamp.br

Prof. Dr. Manuel de Jesus Mendes
UNICAMP / ITI
mendes@dca.fee.unicamp.br
BRAZIL

Workflow management systems are fundamental as a support technology to virtual enterprise partners during its production phase. They support the efficient and automated execution of business process. Although they are an excellent approach when the control flow is specified and implemented from scratch, it has some drawbacks if a workflow management system is to be integrated within already existing proprietary solutions. This paper describes a case study in which a workflow engine and a meta-object based repository are embedded into an existent open service platform with the main objective to verify, in a real world case, the problems and benefits related with the use of embedded workflow engines and meta-object repositories compatible with well-accepted standards like OMG MOF (Meta-Object Facility).

1. INTRODUCTION

Workflow management systems are fundamental as a support technology to virtual enterprise partners during its production phase. They support the efficient and automated execution of dynamic virtual enterprise operations. Although they are an excellent approach when the control flow is specified and implemented from scratch, it has some drawbacks if a workflow management system is to be integrated within an existing solution.

The problem we have here is basically how to change or enhance an existing automation of a business process when even the control flow that coordinates the interoperations among distributed activities may be different from the way a workflow engine is supposed to do and how to aggregate flexibility when some existing environments were not built with flexibility requirements in mind.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35585-6_68

L. M. Camarinha-Matos (ed.), *Collaborative Business Ecosystems and Virtual Enterprises*

This paper describes a case study in which a light-weight workflow engine and a meta-object based repository are embedded in an existent open service platform, like PLATIN platform (Eckert, 2001), with the main objective to verify, in a real world case, the problems and benefits related with the use of embedded light-weight workflow engines and MOF (Meta-Object Facility) repositories (Boldt, 2000).

Next section presents the open service platform in which the engine and the repository will be integrated; In section 3, the process used to create and integrate a MOF repository is presented and finally, we describe the light-weight workflow engine prototype and its applications, as well as, its integration process to the platform.

2. PLATIN PLATFORM

The middleware platform PLATIN forms a processing environment for an open service market into which distributed applications can be managed, launched, and executed. This environment envisaged by TINA (telecommunications information networking architecture) (Chapman, 1995) comprises heterogeneous and interconnected DPE (distributed process environment) nodes that fulfill mainly two purposes: they hide technical and organizational heterogeneity and they present a target system that can help to ease the tasks for application designers.

The middleware platform helps to decouple applications and the systems necessary for their support, such that both can be developed and evolve in different time lines.

In the middleware platform (Eckert, 2001) three layers can be distinguished: Application, DPE and NCCE layers. This division allows technical solutions in each of the problem areas to evolve independently, while making appropriate abstractions used to relate the parts to each other and reasons about their properties. See Figure 1a below:

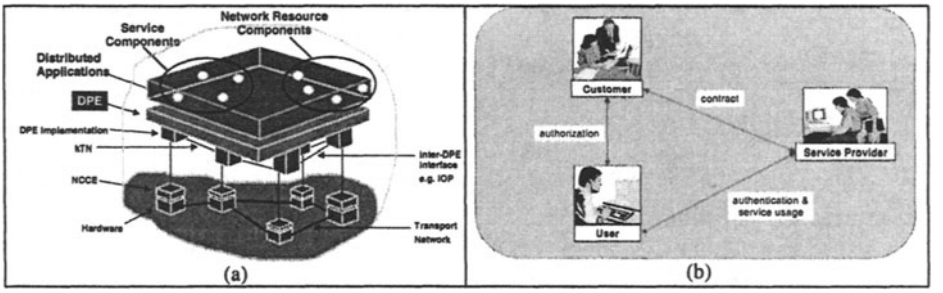

Figure 1 - Stakeholder model of PLATIN (a) and Architectural separations in the middleware platform (b) – (Eckert, 2001)

The Applications layer comprises information and application services being constructed as software components. Applications services offer functionality to human end users, but might even be other applications having an interest in using the offered functionality. Interacting with a particular application is thus using its

service. In addition to distributed applications in general an open service market has the capabilities to be prepared for the commercial situations and value added chaining, i.e. the applications respect the user-customer-provider relationship. See Figure 1b.

The DPE is a layer comprising computing capabilities to access and control the infrastructure and execute software components. The DPE is the infrastructure that software of distributed applications uses as execution environment. This execution is distribution transparent. The DPE consists of nodes, which are interconnected for the purpose of enabling the interoperability between applications and their computational objects installed across different nodes.

The NCCE (native computing and communication environment) layer is a network technology based part, comprising the infrastructure resources. Applications residing in the DPE must be able to access the network transport infrastructure, thus enabling applications to use and control its transport capabilities.

The PLATIN service architecture provides a framework that takes into consideration the requirements of an open service market. It describes how services are developed, offered and provided, and how the mobility of users, output devices and sessions is achieved. Users can access their familiar work environment from different environments, they can use mobile output devices, and they can suspend sessions at any time and resume the session later from a different environment. The stakeholder model depicted in Figure 1b is derived from the service architecture.

The information model describes the different roles of the people and organizations participating in a service market. The retailer and the customer conclude a contract specifying the arrangements for service use (IKV1, 2001).

The workflow engine service and part of the repository code is integrated in the platform as new application services that can be used by other specific application services or by platform clients (users or customers).

3. MOF REPOSITORY

The MOF is an OMG (Object Management Group) standard. It defines a suite of CORBA-based services for managing meta-information. OMG has decided to take MOF as the foundation and reference point for all other facilities inside OMA (Object Management Architecture) and this means that the Workflow Management Facility (Boldt, 2000) also has to integrate into this overall picture. Meta-information is information describing other information. For example, programming language types, database schemas, UML models, etc (Heaton, 2001).

MOF is a core technology to support the development and management of distributed CORBA-based metadata systems. This specification enhances metadata management and metadata interoperability in distributed object environment in general, and in distributed development environment in particular. It also defines a simple meta-meta model with sufficient semantics to describe meta models in various domains starting with the domain of object analysis and design.

Our Repository development process comprises three main steps: formalization of the information model to be used; development of a repository manager and development of a set of repository client applications. The information model used to describe the workflow process definitions and its resources is based in the Nortel

Workflow proposal to the OMG RFP for a workflow management facility (Warne, 1998). The second and third steps are implemented with DSTC dMOF tool product. This tool is, basically, a set of commands with command-line interfaces that give support to the generation of MOF–based repository managers when the information models are defined in MODL (Meta-object Definition Language).

After these steps, the repository manager will interact with its clients through an IDL interface generated by dMOF tool. The main method available to the clients is the object factory for the entire metadata package. Its name is MetaModelPackageFactory. It is the first step to fetch the meta-objects available in the repository.

When the metadata repository has been implemented, it is time to develop the metadata tools needed by the system. These may include: Metadata input/output tools; Metadata interchange tools or Metadata versioning, comparison and archiving tools. In our case, two client programs were necessary. A program to add new process definitions to the repository and a program to use them in run-time to fetch and execute the workflow processes. The first one could be a complex graphical design tool or a simple ad-hoc program without any graphical resources, as we did, just to guarantee that the definitions would be in the repository. The second one is our light-weight workflow engine service implemented as an application service over PLATIN platform. See section 4.

Considering the automatic generation of the repository manager, to integrate the repository in the platform means create a new platform service in the framework to enable the workflow engine and other application services to manage metadata previously stored in a database. The dMOF tool, in version 1.1, enable three alternatives to access a database using JDBC: Oracle, PostgreSQL and MySQL.

PLATIN platform supplies to application developers a framework with several built-in common services (access control, subscription. Accounting, etc) that are important in the context of the open service scenario. See Figure 2:

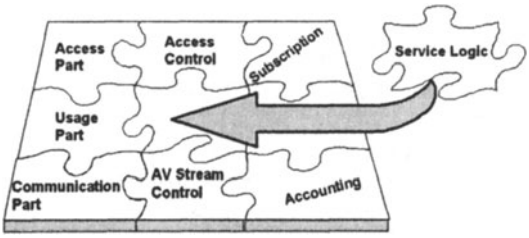

Figure 2 - New Services in PLATIN framework

The new functionality to access the repository was implemented in two parts: a Repository Control Service and a Repository API to be added to PLATIN framework. The first part is an application service with user interface to supply functions as: start the repository, report exceptions, browse its contents and stop the repository operation. The second one is a set of interfaces that can be accessed by developers when a new application service needs to make use of a meta-object repository as the one implemented. The workflow engine is one example of this latter case, it makes use of these interfaces to fetch process definitions.

4. WORKFLOW ENGINE

This section presents the light-weight workflow engine to chain PLATIN application services and a plausible interaction scenario of a fake travel workflow application. The scenario is made up of two fake chained services (hotel reservation and ticket reservation) to be all of them coordinated by a workflow engine that executes a given workflow process definition previously stored in the meta-object repository detailed in section 3. It is a classical example of scheduling and production control inside a production phase of a virtual product life cycle matrix (Bishop, 1993).

The workflow engine and the two fake services are used together in the PLATIN Platform to implement the fake Travel Application. It is made up of two service steps (a first step implemented by the Hotel Reservation Service and the second service step implemented by the Ticket Reservation Service) according to a given service process definition.

Given an appropriate service process definition the actual implementation of the workflow engine is capable of interpreting more complex service process definitions than the correspondent to the fake travel application i.e. applications with parallel and sequence dependencies of any quantity of PLATIN services, provided that the input data and output data between two connected PLATIN services is compatible.

The workflow engine application service implementation (and its use in the fake Travel Application) intends to be just a proof of concept to give some support to the idea that the workflow technology is a suitable means to chain services according to a given service process definition and that it also meets the operational perspectives of the PLATIN Platform, i.e. access session, authentication, profiling. etc. The integration of services into a chain, exploit the information flow between those steps as a further added value.

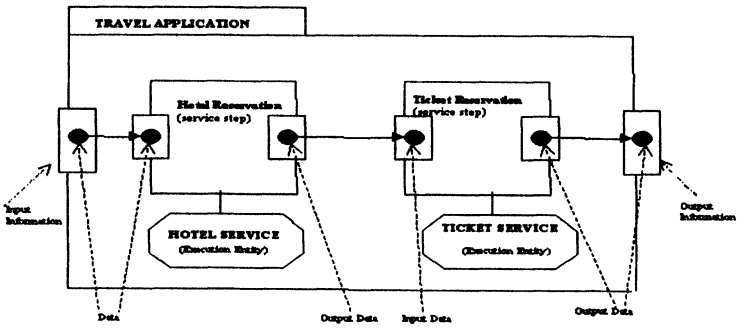

Figure 3 - The Travel Application Definition

At a Build level, the service process definition that corresponds to our fake Travel Application is depicted graphically in Figure 3 with the EDOC notation (Tyndale-Biscoe, 2001) that shows the Travel Application as composed of two service steps: a Hotel Reservation step, a Tickets Reservation step and a set of data dependencies.

The graphical representation shown in Figure 3, has to be translated into an intermediate language (Giordani, 1999) to be computationally interpretable by the workflow engine. For our fake travel application this representation looks like this:

```
Compound_Task( ct_1, Travel Application){
        Input_Set( is_1) { Input_Object( io_1, 'd' ); }
        Output_Set( os_1 ) { Output_Object( oo_1, 'd' ) ; }
Process_Definition() {
        Simple_Task( st_1, Hotel_Reservation, Hotel_Service ) {
                Input_Set( is_1 ) { Input_Object( io_1, 'd' ); }
                Output_Set( os_1 ) { Output_Object( oo_1, 'd' ); } }
        Simple_Task( st_2, Tickets_Reservation, Tickets_Service ) {
                Input_Set( is_1 ) { Input_Object( io_1, 'd' ); }
                Output_Set( os_1 ) { Output_Object( oo_1, 'd' ); } }
Dependency (){
        dep( 1, ct_1.is_1.io_1, st_1.is_1.io_1 );
        dep( 2, st_1.os_1.oo_1, st_2.is_1.io_1 );
        dep( 3, st_2.os_1.oo_1, ct_1.os_1.oo_1 ); } } }.
```

The interpretation of this service process definition is the following. Our service process named 'Travel Application" in Figure 3, is mapped into a **Compound_Task(..)** expression. The expression **Process_Defintion()** is used to describe what it is inside of our service process named **Travel Application**.

Inside this Process_Definition() section, the constituent *service steps* (Hotel_Reservation and Tickets_Reservation) are specified using the parameters of the **Simple_Task (..)** expressions.

Each one separately, the Compound_Task(..) and the Simple_Task expressions, must specify its input and output information using the constructs Input_Set() and OutputSet() and, inside each of them, the actual data to be exchanged between the service steps is specified with Input_Object(..) and Output_Object(..) constructs.

The section Dependency () is used to describe the dependencies among all service steps that conform our service process. Each individual dependency is specified using the expression dep(..) which have three arguments: an identifier of the dependency, the identifier of the source of the information and the identification of the target of the information.

At execution level, the coordination mechanism of the workflow engine when interpreting the service process definition, initially, instantiates a ***Compound Task controller*** object (ct_1 in figure 4) for the whole Travel Application and ***Simple Task Controller*** objects for each one of the service steps that conform it (the hotel reservation step and the tickets reservation, st_1 and st_2 in figure 4). There is a one to one mapping between each service step and a corresponding ***Simple Task Controller*** that controls it.

When the information (Input Object-1) gets to the entry of the ***Compound Task Controller*** (ct_1), an event notification is triggered (using an event service) in the ***Simple Task Controller*** (st_1) that controls the Hotel Reservation step, informing it that its input information(io_1) is available for use. This *Simple Task Controller* (st_1) requests its associated execution entity (Hotel Service) to process the available data. The data produced by the PLATIN Hotel Service is returned to the

Simple Task Controller (st_1) that corresponds to the Hotel service step, as its output data (oo_1).

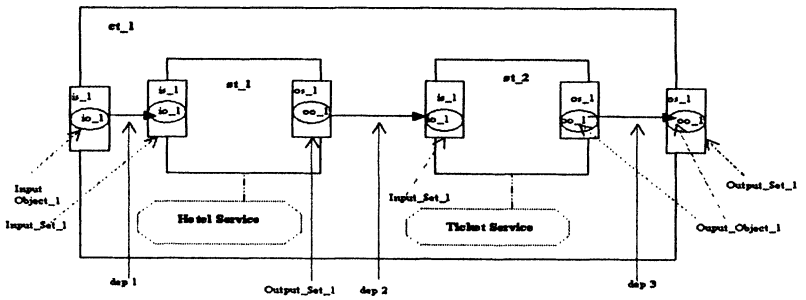

Figure 4 - The Travel Application and the Java classes that implement it

Again, by means of the event service an event notification is triggered in the Simple Task Controller (st_2) of the Tickets Reservation step informing it that its input information (io_1) is available for use (which is the output-oo_1 of the previous Hotel service step). This second Simple Task Controller (st_2) requests its associated execution entity (Tickets Service) to process the available data. The information output produced by the Tickets Service is returned as the output (oo_1) of this second Simple Task Controller (st_2).

After that an event notification is triggered, in this case, in the *Compound Task Controller* that corresponds to the whole Travel Application informing that its output information (oo_1) is available for use. With this the whole Travel Application process ends and the final result is available, to be looked for the PLATIN user that requested the engine application service to chain the Hotel and Ticket services.

As the workflow engine application service receives information from the PLATIN user, and the workflow engine interprets the service process definition, information is sent back and forth between the workflow engine (its constituent Simple Task Controllers), and the execution entities implemented as Platin application services (Hotel Service and Ticket Service).

It means that each PLATIN application service to be chained by a workflow engine must define exactly the format and the semantics of its input information, define exactly the format and the semantics of its output information and clearly define an open interface, with at least three services, so the workflow engine could send to the service its input information, start the service and get the results. In (Muehlen, 2000) classification, we could say that we implement a semi-open solution. That means that external APIs are available to extend the workflow applications but they are not yet standard APIs.

5. CONCLUSIONS

Although each integration process of workflow engines in existing environments is an independent problem, several common problems arise in almost all of them. For

example: how to substitute or enhance an existing control-flow and how to call legacy application services from the workflow engine. Some characteristics of our problem helped us to successfully reach our target: Our light-weight engine was just a basic scheduler to start, control and stop one or more application service components inside the platform and the platform itself was a very suitable environment for the integration, since each application service that has been integrated into the PLATIN Platform could be reached and searched through internal calls to an interface list.

Since our light-weight workflow engine implementation provided only limited features for interacting with execution entities its integration within an existing environment, like the PLATIN Platform, was eased without impacting the existent environment with unnecessary modifications and a longer integration process.

Another important conclusion here is that the use of MOF-based repositories increased the flexibility of the solutions in a very consistent way, since they enabled the modeling of any information as metadata and was extensible enough to aggregate new complex service components in the same environment.

In the future, we plan to extend our work to get further proofs of concepts in areas as: adaptability, application interfaces and modeling.

6. ACKNOWLEDGMENTS

We would like to thank the institute GMD FOKUS, Berlin – Germany, by the opportunity to use their open service platform in this project and by all the support we have received during our work.

Other important help we have had came from Distributed Systems Technology Center – DSTC, Brisbane - Australia. Without the use of dMOF tool, we would have had to implement all the MOF specification or to use a non-standard repository.

7. REFERENCES

1. Eckert K, Körner E, Schoo P. "Introduction to Middleware Platform", PLATIN platform documentation version 1.0, 2001.
2. Boldt, Juergen, "Meta Object Facility Specification" – Version 1.3, formal/00-04-03, 2000.
3. Chapman, Martin, "Overall Concepts and Principles of TINA", 1995.
4. IKV1, "Enago, OSP Version 0.9 Introduction", version 0.9.02, Draft April 2001,
5. Boldt, Juergen, "Workflow Management Specification" – Ver. 1.2, formal/00-05-02, 2000.
6. Heaton, Linda., "Unified Model Specification" – Version 1.4, formal/01-09-71 a 80, 2001.
7. Warne John, "Nortel Revised Submission to the Workflow Management RFP", bom/98-03-01, 1998.
8. DSTC, "dMOF tool User's Guide", release 1.0.1, Distributed Systems Technology Center-DSTC, Brisbane, Australy, 2000.
9. Bishop, A.B., Newman, A.E. et.al. "Designed to work: Production systems and people" Prentice-Hall Inc. New Jersey, 1993.
10. Tyndale-Biscoe, Sandy, "A convenience document, containing the revised EDOC Submission Part I with change marks", ad/01-08-19, 2001.
11. Giordani, Benito, "An Event Oriented Workflow Management System Prototype," Department of Computer Engineering and Industrial Automation, MSc Thesis, Campinas University, Brazil, May 1999.
12. Muehlen MZ, Allen R, "Workflow Classification – Embedded & Autonomous Workflow Management Systems", WfMC, 2000.