Implicit BPM: A Business Process Platform for Transparent Workflow Weaving

Rubén Mondéjar[1,2], Pedro García-López[1], Carles Pairot[1,2], and Enric Brull[2]

[1] Department of Computer Engineering and Maths,
Universitat Rovira i Virgili, Tarragona, Spain
{ruben.mondejar,pedro.garcia,carles.pairot}@urv.cat

[2] Diputació de Tarragona, Spain
enric.brull@dipta.cat

Abstract. The integration of business processes into existing applications involves considerable development efforts and costs for IT departments. This precludes the pervasive implementation of BPM in organizations where important applications remain isolated from the existing workflows.

In this paper, we introduce a novel concept, Workflow Weaving, based on non-intrusive techniques, which achieves transparent integration of business processes into organizational applications. This concept relies on BPM standards, Aspect Oriented Programming, and Web patterns to transparently weave business models among current web applications. A prototype platform is presented, which includes our design of a distributed architecture, and a natural and expressive DSL.

Keywords: Workflow Weaving, Implicit BPM, Distributed Platform, Aspect-Orientation, MVC Architecture.

1 Introduction

There is an increasing demand from organizations to integrate business processes into existing applications. Many of these applications remain apart from the company workflows because they were designed as isolated systems without clear interoperation interfaces. In most cases, the cost of this integration is very high because it implies detailed knowledge of the existing applications, and ad-hoc modifications to provide the required connection with workflow engines. This cost makes it very difficult to adopt BPM strategies in these organizations.

These workflows should provide a way of describing the order of execution and the dependent relationships between the activities of running processes in heterogeneous applications. However, if these processes span existing applications in the organization, their integration implies a costly plumbing and connection development work in every piece of software.

In order to reduce this cost, we introduce a novel concept, namely Workflow Weaving, based on non-intrusive techniques, which achieves transparent integration of business processes into existing web applications. We mainly employ

S. Sadiq, P. Soffer, and H. Völzer (Eds.): BPM 2014, LNCS 8659, pp. 168–183, 2014.
© Springer International Publishing Switzerland 2014

Aspect-Oriented Programming (AOP) [1] to transparently intercept existing web applications and connect them to the workflow system. The novelty of our approach is that we intercept the Model-View-Controller (MVC [2]) pattern in key points in order to avoid a detailed knowledge of the target applications. The MVC pattern enables us to perform black-box [3] wrapping interception and to avoid costly clear-box interception models. The only natural assumption is that any of the intercepted applications must be of an MVC web type.

To simplify application integration, we also provide a Domain Specific Language (DSL [4]) that transparently performs Workflow Weaving. This weaving defines the mechanisms to intercept applications and to inject BPM logic into them. Thus, IT technicians do not need to learn AOP since the simple DSL is responsible for enabling the required interceptors in MVC applications. The major contributions of our approach are:

- Transparent *introspection and interception* of web applications, which benefit from the decoupled nature of the most extended pattern for developing modern web applications (MVC).
- A *natural and expressive* DSL that performs Workflow Weaving by injecting AOP interceptors into web applications. This approach considerably simplifies the integration of business processes into existing applications.
- The design and implementation of a *distributed and implicit* BPM platform, which enables distributed process weaving and management.

The rest of the article is structured as follows. Section 2 shortly introduces the state-of-the-art in BPM integration and implicit techniques fields. In Section 3 we give an overview of our Workflow Weaving technique proposal and its features. In Section 4 we introduce our platform design and implementation. Related work is presented in Section 5, and in Section 6 we draw some conclusions.

2 Background

In this section, we explore relevant background in this area. Firstly, we discuss different topics in the scenarios of BPM integration, including important issues like explicit solutions and support for legacy applications. Secondly, we explore existent techniques to provide transparency and integration concerns.

2.1 BPM Integration

Building systems from the ground up is no longer an acceptable business practice and it is certainly not cost effective. In this setting, Business Process Management (BPM) [5,6] is seen as a mechanism for integrating systems and a way of developing new applications.

Actual BPM solutions are well-known and explicit approaches to implement workflows on top of software applications applicable to a certain business. That approach traditionally supports the separation of the business process from the core application, but presents important drawbacks. Some of these disadvantages

include the accommodation of transversal business processes into applications, the combination of different design and execution environments, and the fact of dealing with legacy applications. For these reasons, BPM is in many cases perceived as being expensive and really complex to deploy.

Unanticipated business processes that need to be modelled and incorporated into any operating applications are a common requirement [7,8,9] to accommodate any change in policies, regulations, etc. In addition, business processes should also be easily reused among a variety of applications between the same organization. Such requirements are usually deemed to be painful because existing solutions use explicit techniques.

Since business processes are designed by business analysts, these need to be defined and understood by stakeholders, and they are not typically adept in application development [10]. In this line, the business process must be defined using a high-level domain language, thus hiding technical concerns. As a consequence, business processes are implemented combining standard software engineering approaches, such as object-oriented programming languages (e.g. Java), description languages (e.g. XML), and high-level domain languages (e.g. BPMN).

Finally, another important issue is how to deal with existing legacy applications. Since understanding existing legacy code through reengineering is a challenging task that may consume a lot of resources. Some recent works [11] propose to rewrite them using BPM. Unfortunately, building systems from the ground up can also represent an enormous cost. As an alternative [12] presents a reengineering tool to identify business rules contained in legacy source code. But as authors explain, reengineering using BPM is not easy to apply, because there are no tools that help developers understand the legacy system behaviour.

2.2 Implicit Techniques

Different approaches are taken on implicit middleware [13], like generic wrapping techniques which are normally more intrusive, as well as ad-hoc interception solutions [14] provided by a specific platform in an explicit way.

However, in order to solve transparency or genericity limitations, we can use powerful interception solutions like AOP, which is an established paradigm. Indeed, it enables describing and separating crosscutting system concerns in a modular and highly reusable manner. AOP supports switching on and off new behaviours at a specific point of program execution, while maintaining the system well modularized.

AOP applies to support flexibility and adaptability of applications/services by allowing to switch on and off orthogonal functions, allowing less interdependence and more transparency. The interception is performed in a *join point* (a point in the execution flow), and defined inside a *pointcut* (a set of join points). Whenever the application execution reaches one pointcut, an *advice* (namely a callback) associated with it is executed. The aspect is a module encapsulating pointcuts and advices. It specifies the new functionality to be included and the place in the execution of the original code where this functionality is to be inserted.

In this setting, a weaver is the AOP mechanism that combines code encapsulated in aspects with the original code. There are different weaving mechanisms that can be classified as static or dynamic. Dynamic weaving enables the interchangeability or deactivation of aspects during program execution, while static weaving disallows such capability, i.e. once defined, aspects cannot be deactivated or exchanged.

Finally, we can distinguish between clear-box and black-box approaches [3] to AOP. Clear-box approaches to AOP examine the program internals and source code, producing a combination of program and aspects. Black-box approaches shroud components with aspect wrappers in strategic points avoiding a detailed knowledge of the code internals. Obviously, clear-box or white-box approaches to interception imply more cost and they are more difficult to apply in real settings. Black-box or wrapper-based techniques [15] can considerably simplify the distributed interception [16] of existing systems.

3 Workflow Weaving

Commonly, software applications are developed to automate and to make efficient business processes, which are previously modelled by analysts. Their requirements are functional and represent activities that the organization is currently trying to achieve. However, once applications are finally released, functional requirements inevitably and naturally change in a major or minor degree, evolving to their clients desires and thus improving their functionalities.

In this section, we introduce a novel technique named Workflow Weaving, that enables integration of business processes, represented by BPMN models, like true crosscutting concerns into corporate web applications. Such technique allows integrating business processes with heterogeneous web applications transparently. In this setting, transparent means that our solution must avoid access, modification and detailed knowledge of the source code of the existing applications. The major requirements of our so-called transparent integration are:

- A generic code interception of modern web applications using a black-box solution (Section 3.2). In addition, it must provide introspection capabilities that offer information about models, controllers, and views in the existing web applications to be integrated.
- An easy management and deployment of interceptor code. This requires code injection using a common interface (Section 3.3), where IT technicians are unaware of the application code.
- A high-level domain language and interpreter (Section 3.4) simplifying the integration of business processes and web applications. This avoids knowledge of the underlying interception framework (AOP).

The rest of the section explains how the Workflow Weaving technique deals with each of these requirements itemized above.

3.1 Use Case

Clear examples of real application requirements, are authentication portals or payment gateways, which use web redirections to change the navigation rules and other behaviours of the system.

In Figure 1 we present a use case based on two applications within the same organization: an e-commerce Pet Store application, and a generic Accounting application. The Pet Store is a classic sample application from the Java EE Platform, used to show its features. We have also implemented a generic Accounting application that manages the books, and the customers of an enterprise.

Moreover, a business analyst has designed and modelled a Purchase Workflow in this scenario, limiting itself to a standard BPMN 2.0 design tool. Note that since this is a simple example, the represented tasks are user tasks, although they can be of another type, because our plaftorm provides a full support of BPMN 2.0 activities.

Both applications are implemented on a MVC framework, and their components are: **models**, consisting of persisted domain objects, **controllers**, formed by a set of action to command interactions, and **view** pages, which communicate directly with the end-users. For their graphical representation, we use the UML notation [17] that illustrates interactions among the MVC components of a web application. In addition, the UML notation has the following basic rules: view pages can only interact with controllers, model objects can only interact with controllers, and controllers can interact with any component.

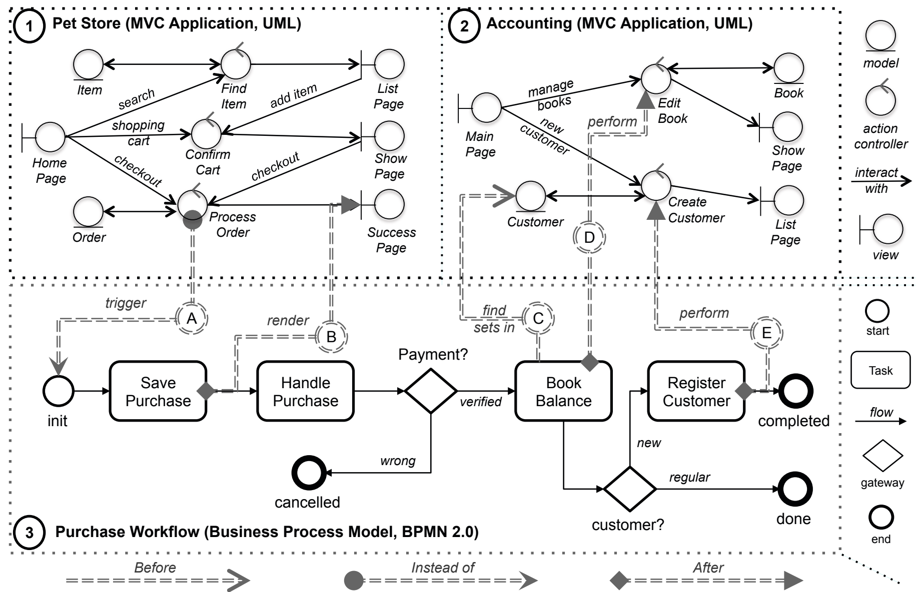

Fig. 1. Workflow Weaving Use Case

Lastly, to interrelate the different diagrams, we use dashed arrows to indicate the existent Workflow Weaving among the business process and the application. Note that each arrow has a tag describing the associated action, and there are three different arrow shapes depending on the interception type: **before**, **instead of**, and **after**.

In Figure 1 we have highlighted the most important spots where Workflow Weaving occurs:

(A) in the PetStore application, **instead of** the *Process* action from the *Order* controller, the *init* event of the Purchase Workflow is triggered.

(B) **After** the *Save Purchase* task is completed, the execution returns to the application to render the *Success Page* view.

Later on, the Purchase Workflow continues its natural execution, until we arrive to the *Book Balance* task. This means that some other participant has claimed and completed the *Handle Purchase* task.

(C) Once the payment is verified, the execution flow moves to the Accounting application. Particularly, **before** the *Book Balance* task is started, the process looks up the *customer* model by its National Identification Number (*NIN*) and sets the result into the *exists* boolean attribute of this task.

(D) **After** the *Book Balance* task is completed, the weaver performs the *Edit* action from the *Book* controller.

(E) Lastly, **after** the *Register Customer* task, the weaver performs the *Create* action from the *Customer* controller.

As seen on the example, the Workflow Weaving technique defines the whole behaviour of the system when the process is running. In the next section, we present how our technique can crosscut MVC applications transparently, providing a true black-box solution.

3.2 MVC Pattern

Web development has changed significantly over the past few years. It has not been long since deploying a web project simply involved uploading static HTML, CSS and JavaScript files to a web server. Nowadays, web application development using web frameworks has become the *de facto* work environment. Furthermore, current frameworks (e.g. Grails [18]) follow common fundamentals and best practise principles like reutilization (i.e. DRY - Don't Repeat Yourself) or productivity (e.g., conventions over configurations, and scaffolding).

Furthermore, the most important of the common features on modern web frameworks is the whole adoption of the MVC pattern. Although MVC was originally developed for desktop computing, it has been widely adopted as an architecture for web applications in all major programming languages. As a result, new MVC frameworks have appeared that provide structure and guidance when developing these applications.

In this work, we mainly focus on MVC based applications, which are mainly those based on modern web frameworks. This restriction allows us to provide a real black-box solution. Thus, even though the entirety of legacy applications

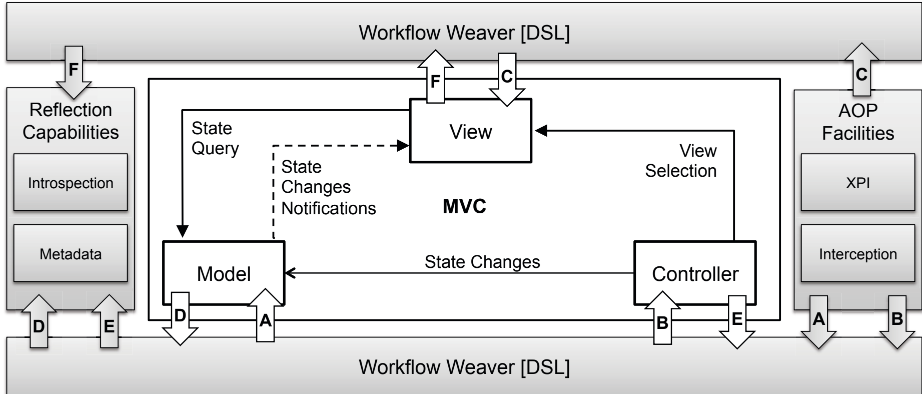

Fig. 2. Black-Box Diagram for the MVC Pattern

may not be included, they can be refactored following some guidelines, like those presented in other works [2].

In our case, we introduce a solution that follows a black-box model injecting code in strategic points of the web application framework. Thanks to the MVC standard pattern, which is used extensively in web frameworks, we are able to intercept models, views and controllers in a transparent and decoupled way. Thus, as we have shown in Figure 2, the MVC pattern enables the use of AOP facilities to intercept code (in points A, B, and C), and to use reflection and introspection techniques to obtain the necessary information (in points D, E, and F).

Indeed, reflection is a well-known self-management technique for providing mechanisms to inspect a system structure and behaviour. Furthermore, the MVC pattern presents a clear facade, standard naming conventions, and inheritance rules to easily perform automatic introspection of the application. In this case, we benefit from this advantage to extract the declared model and attributes (A), the enabled controllers and actions (B), and the deployed views and navigation rules (C) among all of them. Additionally, the workflow weaver also injects new code to add or remove calculated fields to domain classes (D), to change the current behaviour of the controller actions (E), or inject code in compiled versions of the page views (F).

Accordingly, our solution benefits from the MVC pattern to become a generic solution that fulfils the black-box requirement. Furthermore, the limitation of Workflow Weaving to MVC based web applications is not a big constraint, given that MVC is the most common architecture pattern.

3.3 Crosscutting Interfaces

Crosscut Programming Interfaces (XPI) [19] are explicit, abstract interfaces that provide a clear separation between the interceptor logic and the AOP language or

```
1:  public abstract aspect MvcXpi {
2:      public pointcut inController(): within(*..*Controller);
3:      public pointcut controllerAction(): inController() && execution(@Action public * *.*(..));
4:      public pointcut inModel(): within(*.persistence.Entity);
5:      public pointcut saveMethod(): inModel() && execution(public Object save(..));
6:      //More CRUD Methods (...)
7:      public pointcut inView(): within(*..gsp_*_gsp);
8:  }
```

Fig. 3. XPI Example for a MVC Framework

implementation. It allows for their separate and parallel evolution and produces a better correspondence between programs and designs.

In our scenario, each modern web framework uses different implementations and mechanisms to instantiate domain classes, inject controllers, handle the data layer, represent the views, among others. In fact, each of them presents different approaches to implement the MVC pattern and its entities, using different paradigms like object-oriented inheritance, XML configuration, or code annotations.

The idea of the XPI is to create a contract between the platform and the intercepted system. Therefore, the XPI establishes a binding with the MVC Framework which states the pointcuts definition; and it also establishes another binding with the interception platform, namely the advice method definitions. As a result, if the platform uses different XPIs without modifying the advice method definitions, each application is able to be implemented in any MVC framework.

For instance, in Figure 3, we present a crosscutting interface for intercepting each action method, and the persistence model CRUD methods. This example intercepts a specific MVC framework (i.e., Grails [18]) where Controllers follow a name convention (e.g., CartController) and Model domain classes (e.g., Cart) use inheritance from a persistence entity class. Therefore, with a simple but effective variation in the pointcut definition, for instance, intercepting those Controllers marked with a common annotation (e.g., *@Controller*), we obtain an XPI suitable to intercept another MVC framework.

In addition to this introspection solution, we also need to use the previously explained reflection capabilities to properly extract the system metadata. Thus, we extend this API with the methods to obtain each instance living in the system: models, controllers, among others. For this purpose, we implement interception methods that store these object instances, at model construction and controller injection time.

As a conclusion, we can state that although each MVC implementation requires its own pointcuts, the XPI allows us to maintain the necessary separation between our solution and the framework particularities. Note that this kind of solution allows the platform to intercept at the same time different applications implemented with different MVC frameworks. Therefore, our proposal benefits from the XPI approach to be the abstracted solution that fulfils the second requirement of this section.

```
 1:  dsl = name '{' {weaver} '}' ;
 2:  weaver = in application ':' {act ',' behaviour} ';' ;
 3:  application = name ;
 4   act = when variable element [from controller];
 5:  when = Before | Instead of | After ;
 6:  element = action | view | event | task | attribute | flow ;
 7:  controller = name ;
 8:  behaviour = connector variable element [from controller] [by variable] [{another}] ;
 9:  connector = perform | find | save | render | trigger | start | sets in ;
10:  another = and behaviour ;
11:  name = { all characters - '"' } ;
12:  variable = '"' , name , '"' ;
13:  all characters = ? all visible characters ? ;
```

Fig. 4. Reduced DSL Grammar

3.4 DSL

It is well known that AOP paradigm has not been adopted by developers and organizations due to its inherent complexity [20]. On the other hand, a Domain Specific Language (DSL) is a reduced language whose main aim is to represent constructions for a given domain. To begin with, a simple and understandable human readable language is required. Thereby, if we are able to provide an adequate DSL, end IT technicians do not need to deal with the underlying AOP facilities.

For our approach, we propose a DSL specification (Figure 4), which provides the way to formalize an abstract descriptor for the Workflow Weaving technique. Basically, this DSL specifies the Workflow Weaving behaviours, and all the interactions among each element in the system (i.e., applications and process model).

The definition of each *dsl* starts with the *name*, which obviously has to be unique in the system once it is deployed, and needs to be the same as the class name (e.g., weaver.PurchaseProcess). We continue with the *workflow-weaver* list. Each workflow weaver determines the target application, and its collection of events, as well as their related behaviours.

In line 4 (Figure 4), the *act* construction is defined, thus establishing when (past, present or future) and how (i.e., basically which is the involved *element*) it is produced in the specified application. Finally, if it is an action, we can specify from which *controller* it comes from. Moreover, in this DSL we specify each *behaviour* bound to an *act*, and it is defined in a similar manner. The *connector* introduces the action that we want to execute (e.g., *render*), and the *element* that will receive it.

Indeed, connectors are the major hook points that bind business processes and MVC web applications. In this line, we have defined a basic set of connectors that allow IT technicians to **start** an event or **trigger** a task from a business processes, access or modify (**find or save**) domain objects from the model, **render** view pages, and **perform** action methods of a controller.

In Figure 5 we show an example of DSL based on the use case described in Section 3.1. This DSL example defines a Workflow Weaving among the *Purchase*

```
1:  PurchaseWorkflow {
2:     in PetStore :
3:        Instead of "process" action from Order, trigger "init" event;
4:        After "Save Purchase" task, render "success" view;
5:     in Accounting :
6:        Before "Book Balance" task, find "customer" by "nin" and sets in "exists" attribute;
7:        After "Book Balance" task, perform "update" action from Book;
8:        After "Register Customer" task, perform "create" action from Customer;
9:  }
```

Fig. 5. Purchase Workflow DSL Example

process and the *PetStore* and *Accounting* applications. As we can see, each workflow weaver is defined simply by following the dotted arrows and the model designed by the business analyst.

Lastly, although this DSL grammar is flexible enough for basic model interactions, in the future we will extend it for executing new actions or accordingly capturing other events with platform implementation functionalities.

Indeed, this use case example demonstrates the usability and expressiveness of our approach. We have shown that our proposal builds a DSL solution including a high-level domain language, which considerably abstracts the use of AOP. Therefore, the main benefit is that the IT department does not need experts in the AOP field.

4 Implicit BPM Approach

In order to materialize the Workflow Weaving technique we have implemented the Implicit BPM approach. We have designed it as a simple, decoupled, and distributed platform. In this section, we introduce our Implicit BPM platform architecture and its life-cycle.

4.1 Architecture

The Implicit BPM platform has already been implemented as a distributed architecture, which consists of two separate parts: the Front-End and the Back-End systems. Both parts of the platform are connected via web standard mechanisms for flexibility and extensibility reasons, as well as due to the suitability of the web paradigm for exposing and consuming remote services.

We can see a diagram of our architecture proposal in Figure 6, where grey coloured components are either newly implemented or extended for our approach and are discussed as follows.

Front-End Side: is where organizational applications are deployed and where they run on servers in a distributed way. In fact, this system provides the interception and reflection components in our MVC black-box solution (as we have shown in Figure 2).

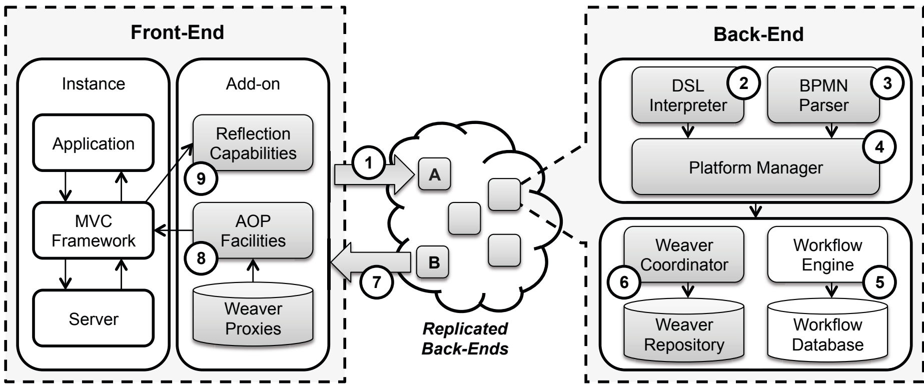

Fig. 6. Implict BPM Platform Architecture

Reflection Capabilities: include all the introspection functionalities. Specifically, this reflective approach retrieves meta-data from the MVC application, for example, the relationship between each model, controller, and view.

Weaver Proxies: live in a container that provides inserting (i.e., deployment) and installing (i.e., activation) them into the distributed and registered applications. In the deployment phase, the weaver proxy is sent to the Front-End container in the same host where the targeted application is running. Whenever the weaver proxy is needed, it has to be loaded from the container (i.e., serialized class) into the application.

AOP Facilities: support and use interceptors intensively, and strictly follow the XPI defined for this purpose. Algorithm 1 shows a workflow weaver example, which is responsible to render a view from a task, after a specific action is performed.

Algorithm 1. *renderTaskView*
Advice: *afterAction /* Pointcut */*
Input: *resp /* Response */*
Output: *resp /* Response */*

1: $acts \leftarrow proxy.getActs()$
2: $action \leftarrow joinPoint.thisMethodName()$
3: **if** $action \in act.getActions()$ **then**
4: $\quad act \leftarrow proxy.getAct(action)$
5: $\quad behaviours \leftarrow proxy.getBehaviours(action)$
6: $\quad$ **for** $beh \in behaviours$ **do**
7: $\quad\quad$ **if** $beh.connector = render$ **then**
8: $\quad\quad\quad attrs \leftarrow backEnd.getTaskAttrs(act.task)$
9: $\quad\quad\quad resp \leftarrow proxy.render(attrs)$
10: $\quad\quad$ **end if**
11: $\quad$ **end for**
12: **end if**
13: $return(resp)$

For this purpose, this algorithm intercepts a specified action of a controller, and changes the application behaviour just following the enabled DSL instructions. In addition, it is able to recover the application metadata from the local proxy, and the model information (e.g. task attributes) from remote Back-End instances. We can see an example of this algorithm injection in our use case (Figure 1, Point B) and its DSL code (Figure 5, Line 4) where the Front-End detects the act *After 'Save Purchase' task*, and consequently execute the behaviour *render 'success' view*.

This way, each activated workflow weaver is converted to an AOP interceptor and is loaded in the proper proxy instance. With the XPI, the dynamic and decoupled techniques of the Weaver Proxy produce an important benefit we already explained in the previous section: it supports runtime reconfiguration, although the underlying weaver, from the AOP facilities, does not.

Back-End Side: supports the infrastructure of the system and provides deployment, management, and execution capabilities.

BPMN Parser: transforms the uploaded BPMN 2.0 XML file into a Workflow Engine component. Previously, it verifies the correcteness of the model and it saves the necessary data into the platform to allow future Workflow Weavers to be integrated.

DSL Interpreter: is included in the Back-End to allow developers to write accurate DSL codes. It takes advantage of the remote MVC instrospection to facilitate the creation and edition of the Workflow Weaver. Using the communication system, it remotely retrieves all the necessary information from the Front-End reflection capabilities.

Platform Manager: controls, coordinates and consolidates the BPMN models, DSL codes, and registered applications. As a first layer of functionality for the Back-End, administrators are able to deploy models and classes, and manage the registered applications. Finally, it also provides reporting features for the business analysts.

Weaver Coordinator: is responsible for handling the state of each distributed workflow weaver in the platform. In addition, it periodically receives information about the deployed and activated weavers to perform monitoring tasks into the platform.

Weaver Repository: is capable of storing wokflow-weavers remotely. These weavers need to scatter the DSL parts and link them to each business process, which was previously deployed into the Workflow Engine.

Replicated Back-Ends: extend scalability features on this platform side. Application instances are duplicated, since they are mirror images of each other, and running on multiple servers of the organization.

Communication Bus: uses the standard HTTP protocol to expose all the services between the Front-End and the Back-End systems, mainly via REST technology.

4.2 Platform Life Cycle

Figure 6 shows the platform life cycle. First of all, enabling the **Front-End Add-on** an any application automatically registers it (1) into the **Back-End (A) instance**. Once an application is redeployed, the same process is executed in order to detect possible updates, for instance, a simple modification in the application deployed URL. Using remote reflection capabilities, the platform is able to retrieve all the needed information, as the application name, the operating URL, the main domain class (i.e., process instance binding) and the MVC structure and navigation rules.

Later on, in order to perform a new deployment, an administrator user needs: a **DSL** code class (2), and the related **BPMN** model (3) exported from a workflow designer. The **Platform Manager** (4) checks the consistency and coherency between the deployed files, and determines if it can be formally introduced into the system. If it is successfully installed, the generated classes go to the underlying systems (5-6), saving them in their supported **Database** and **Repository**, respectively.

The **Workflow Engine** (5) is wrapped by the **Back-End** system and it is used like a set of services which are exposed in a distributed fashion. Therefore, the engine is completely decoupled from the platform, and it could be easily switched by another BPM compliance implementation (e.g., Activiti[21] or Camunda[22]). In this scenario, tasks are executed within targeted applications, even the service tasks, that have to reference and use code from its corresponding application. Other types of activities, like manual tasks, do not have any special requirements.

Next, the **Weaver Coordinator** (6) controls all remote instances. Following the example in Figure 6, the **Back-End (B) instance** directly (7) deploys, and later injects the specified weaver proxies using the remote **AOP Facilities** (8).

In the end, these **Weaver Proxies** live in each application and they follows the instructions from the deployed DSL. To perform these rules, **Weaver Proxies** use the necessary AOP mechanisms implemented into the **Front-End Add-on** (e.g., Algorithm 1), as a black-box solution.

Finally, since the black-box mechanisms are linked to the XPI contract, they can use **Reflection Capabilities** (9) to return gathered feedback information (1) to any Back-End instance. This information is sent to the **Weaver Coordinator** (6) for monitoring purposes, as well as for reporting tasks to the **Platform Manager** (4).

5 Related Work

Previous works in literature [7,8,9] used AOP interception techniques to integrate applications with BPM platforms or external rule engines. For example, [7] proposed hybrid aspects for integrating object-oriented programming applications and rule-based reasoning. However, this approach relies on the in-depth knowledge of the targeted application to deploy the appropriate interceptors and pointcuts. The adoption of DSLs to simplify the implementation of interceptors

have been also proposed in the past [9]. The main disadvantage of these DSLs is that to successfully apply them: they need to know the business classes, relationships among them, the semantics of their methods, and the interactions among instances.

Another related work is [23], where authors propose an AOP approach to separate out the base workflow from addition workflows, which can be weaved into the base when additional features are selected. Again, these works require detailed knowledge of the application that must be intercepted.

Interception approaches for standards such as BPEL have also been proposed before. AO4BPEL [24] and BPEL'n'aspects [25], are specific aspect-oriented language extensions. Each implementation is based on a modified BPEL engine, which checks at all potential join points, if an aspect has specified it in its pointcut. This allows easy and dynamic weaving of BPEL aspects with the drawback of less performance. As we plan to implement generic AOP mechanisms, we will not change the workflow engine but perform weaving on model level prior to workflow execution.

The major difference between all aforementioned previous works is that they follow a clear-box AOP interception model that requires in-depth knowledge of the application that must be intercepted. This clear-box model clearly complicates the adoption of these approaches and reduces their potential uses. In our case, we follow a black-box approach that intercepts code in strategic points using AOP facilities and following a common XPI for MVC frameworks. Thanks to the MVC standard pattern used extensively in web frameworks, we are able to interrogate and manipulate models, views and controllers transparently to the internal code of each application.

As stated before, reengineering legacy applications using BPM [12] is not easy to apply, because there are no tools that help the developers understand the legacy system behaviour. The introspection and wrapping capabilities on top of the MVC pattern enable us to perform black-box interception of any web application using this pattern. This considerably simplifies the integration of Web applications with BPM platforms using our DSL. The users of our DSL do not need to study the code of the existing application, and can thus weave business processes in legacy applications.

We propose the first approach of a distributed platform that interconnects different MVC based applications, and which allows business analysts to observe, proceduralize, and model each process in a holistic way.

6 Conclusions

We outline the importance of integrating business processes into existing applications. Nevertheless, rewriting legacy applications or reengineering them for BPM integration involves important development costs and in-depth knowledge of the targeted applications.

In this paper we present a novel solution (Implicit BPM) for integrating business processes into existing core applications as if they were a whole system.

We introduce a new concept, namely Workflow Weaving, based on non-intrusive techniques, which achieves this kind of integration transparently.

The novelty of our approach is to use black-box AOP techniques that benefit from the MVC web pattern to weave processes in a more transparent way. Previous approaches in literature used clear-box models, which require detailed knowledge of the legacy application. In our previous works [16,26] based on the same underlying distributed AOP principles, we have accurately evaluated that this kind of approach does not impose an additional overhead.

We also provide a natural and easy to use DSL that considerably simplifies the workflow weaving process, while at the same time hiding the underlying AOP complexity. Our prototype implementation is freely available at `http://implicit-bpm.sf.net`, under a LGPL license. This prototype makes use of the following well-known and widespread technologies: Groovy, Activiti, AspectJ, and Grails. This implementation includes the Front-End Add-on, which includes the MVC weaver, the Back-End system, which contains the DSL interpreter, and the Platform Manager, as well as the mentioned use case applications.

Cloud computing will enable organizations to bypass expensive BPM Enterprise products and start using open BPM platform solutions into their own private clouds. For these, we have designed and implemented our approach to be perfectly suitable for easy deployment and operation into a Cloud. In a close future, we are going to combine our platform with a private PaaS Cloud, like CloudSNAP [26].

Acknowledgments. We thank the BPM chairs and the three anonymous reviewers for their constructive comments, which helped us to improve this work. In addition, we also want to thank to Manuel Bertran for his many helpful review and suggestions.

This work has been partially funded by the EU in the context of the project CloudSpaces: Open Service Platform for the Next Generation of Personal Clouds (FP7- 317555).

References

1. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., Irwin, J.: Aspect-oriented programming. In: Akşit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
2. Ping, Y., Kontogiannis, K., Lau, T.C.: Transforming legacy web applications to the mvc architecture. In: STEP, Washington, USA, pp. 133–142 (2003)
3. Elrad, T., Filman, R.E., Bader, A.: Aspect-oriented programming: Introduction. Communications of the ACM 44, 29–32 (2001)
4. Dinkelaker, T., Eichberg, M., Mezini, M.: An Architecture for Composing Embedded Domain-specific Languages. In: AOSD, pp. 49–60 (2010)
5. Jablonski, S.: A Software Architecture for Workflow Management Systems. In: DESA, pp. 739–744. IEEE Computer Society (1998)

6. Knuplesch, D., Reichert, M., Fdhila, W., Rinderle-Ma, S.: On enabling compliance of cross-organizational business processes. In: Daniel, F., Wang, J., Weber, B. (eds.) BPM 2013. LNCS, vol. 8094, pp. 146–154. Springer, Heidelberg (2013)
7. D'Hondt, M., Jonckers, V.: Hybrid Aspects for Weaving Object-oriented Functionality and Rule-based knowledge. In: AOSD, pp. 132–140 (2004)
8. Cibran, M., D'hondt, M.: High-Level Specification of Business Rules and Their Crosscutting Connections. In: AOSD (2006)
9. Hnatkowska, B., Kasprzyk, K.: Integration of application business logic and business rules with DSL and AOP. In: Szmuc, T., Szpyrka, M., Zendulka, J. (eds.) CEE-SET 2009. LNCS, vol. 7054, pp. 30–39. Springer, Heidelberg (2012)
10. Geiger, M., Wirtz, G.: Detecting Interoperability and Correctness Issues in BPMN 2.0 Process Models. ZEUS, Rostock, Germany (2013)
11. do Nascimento, G.S., Iochpe, C., Thom, L.H., Reichert, M.: A Method for Rewriting Legacy Systems using Business Process Management Technology. In: ICEIS (3), pp. 57–62 (2009)
12. do Nascimento, G.S., Iochpe, C., Thom, L., Kalsing, A.C., Moreira, Á.: Identifying Business Rules to Legacy Systems Reengineering Based on BPM and SOA. In: ICCSA, pp. 67–82 (2011)
13. Patel, S.R., Gerald, B., Micah, S.: Mastering Enterprise JavaBeans 3.0. John Wiley & Sons, Inc., New York (2006)
14. Schmidt, D., Stal, M., Rohnert, H., Buschmann, F.: Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects, vol. 2. John Wiley & Sons (2000)
15. Mondéjar, R., García-López, P., Fernández-Casado, E., Pairot, C.: TaKo: Providing transparent collaboration on single-user applications. Computer Languages, Systems & Structures 38, 108–121 (2012)
16. Mondejar, R., Garcia-Lopez, P., Pairot, C., Pamies-Juarez, L.: Damon: a Distributed AOP Middleware for Large-Scale Scenarios. Information and Software Technology 54, 317–330 (2012)
17. Rosenberg, D., Scott, K., Matter, F.: Use Case Driven Object Modeling with UML: A Practical Approach (1999)
18. Rocher, G.K., Brown, J., Laforge, G.: The Definitive Guide to Grails. Springer (2009)
19. Griswold, W.G., Sullivan, K., Song, Y., Shonle, M., Tewari, N.: Modular Software Design with Crosscutting Interfaces. IEEE Software 23, 51–60 (2006)
20. Hohenstein, U.D.C., Jäger, M.C.: Using aspect-orientation in industrial projects: Appreciated or damned? In: AOSD, pp. 213–222 (2009)
21. Rademakers, T.: Activiti in Action: Executable business processes in BPMN 2.0. Manning Publications Co. (2012)
22. Freund, J., Rücker, B.: Real-Life BPMN: Using BPMN 2.0 to Analyze, Improve, and Automate Processes in Your Company (2012)
23. Elsner, C.: Towards separation of concerns in model transformation workflows. In: EA, pp. 81–88 (2008)
24. Charfi, A., Mezini, M.: Ao4bpel: An aspect-oriented extension to BPEL. World Wide Web 10, 309–344 (2007)
25. Sonntag, M., Karastoyanova, D.: Compensation of adapted service orchestration logic in bPEL'n'Aspects. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 413–428. Springer, Heidelberg (2011)
26. Mondéjar, R., García-López, P., Pairot, C., Pamies-Juarez, L.: CloudSNAP: A transparent infrastructure for decentralized web deployment using distributed interception. Future Generation Computer Systems 29, 370–380 (2013)