

# Stateless Process Enactment

Raf Haesen<sup>1,2</sup>, Lotte De Rore<sup>1</sup>, Stijn Goedertier<sup>1</sup>, Monique Snoeck<sup>1</sup>, Wilfried Lemahieu<sup>1</sup>, Stephan Poelmans<sup>2</sup>

<sup>1</sup> LIRIS Group, Faculty of Economics and Applied Economics, K.U.Leuven  
Naamsestraat 69, 3000 Leuven, Belgium  
firstName.lastName@econ.kuleuven.be

<sup>2</sup> Campus Vlekho  
Koningsstraat 336, 1030 Brussel, Belgium  
firstName.lastName@vlekho.wenk.be

## **Name**

Stateless Process Enactment

## **Audience**

Business process designers, Process-Aware Information Systems designers

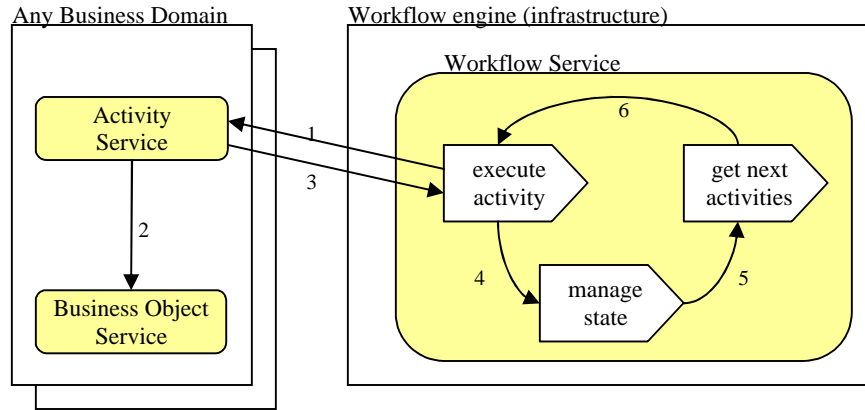
## **Context**

An average company contains many information systems to support the daily work of its employees. A set of database and mainframe transactions enable the consistent retrieval and modification of business data. On top of those transactions, applications are offered that support the execution of clearly delineated activities, such as opening a new claim case, registering damage, fraud detection or indemnity payment.

A business process represents the coordination of a set of activities to achieve a higher-level business goal. For example, a claim handling process coordinates different activities to process insurance claims. The coordination of these activities can be automated by means of a workflow engine that interprets process descriptions and manages the accompanying process instances.

Figure 1 sketches a service-oriented enterprise architecture, consisting of a number of business domains and a workflow engine. A business domain offers business object services and activity services, while a workflow engine offers workflow services:

- *Business object services* provide access to business data and associated business logic in a transactional way.
- *Activity services* support the execution of activities to be executed by human actors or machines. An activity service can use one or more business object services.
- *Workflow services* coordinate activity services according to a business process.

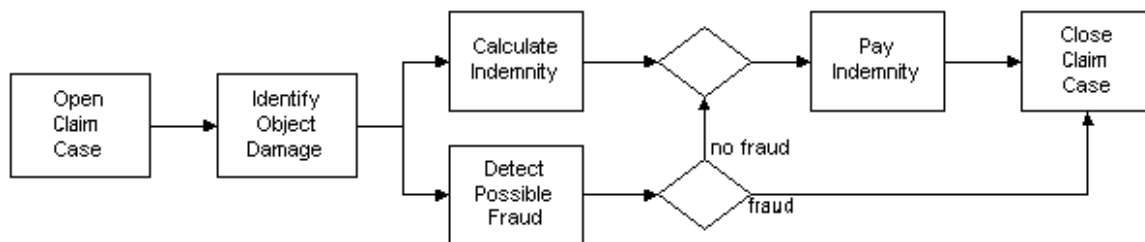


**Figure 1 – Service-Oriented Enterprise Architecture**

Figure 1 also indicates some relevant steps of process enactment: the workflow engine typically guides the actor to a particular activity service (step 1). Eventually one or more business object services will change a part of the business data (step 2). After execution of the activity, the workflow service continues the enactment of the process (step 3). After the workflow service has assessed the state of the process (step 4) the next activities can be determined (step 5) and executed (step 6).

## **Problem**

Figure 2 shows a typical representation of a business process, using the Business Process Modeling Notation (BPMN). Since all activities and their routing are explicitly predefined, state can be maintained in an efficient way, as it only involves updating one or more points of execution. The example represents a simplified version of a claim handling process with the sole focus on the control flow between the activities. After the case is opened, a claim handling expert needs to assess the damage of the involved object. Subsequently, the indemnity can be calculated in parallel with the investigation for a possible fraudulent case. If fraud is detected, the case is immediately closed; otherwise the indemnity is paid and finally the claim case is closed.



**Figure 2 – Claim handling process**

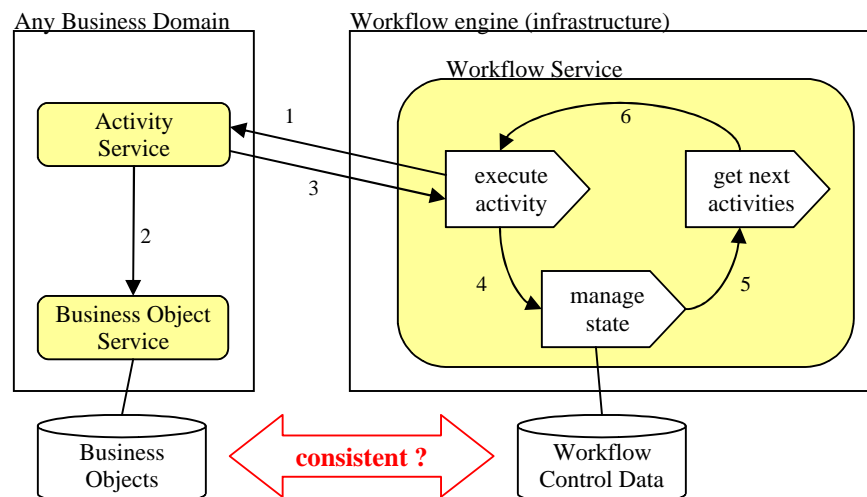
Most existing workflow engines explicitly store the state of running process instances as part of its workflow control data [1]. Such a workflow engine is called **stateful** since the

enabling (pre-condition) and completion (post-condition) of activities is explicitly maintained in the engine for each activity of the process.

Stateful enactment boils down to the replication of a business object's state into the workflow engine entailing all known problems associated with replication of data such as synchronization problems. The main disadvantage of this stateful setup is that the process state will be invalid if the workflow engine is bypassed or if activity execution and state modification form no single unit of work. The following two scenarios can occur and will cause inconsistencies between the data in the business domain and the workflow control data (see Figure 3):

1. Experts should not be forced to follow predefined processes since the obligation to follow a predefined path of activities might be counterproductive. For example, a senior claim handler with many years of experience knows which activities he has to perform to handle a claim without the assistance of a workflow engine. As a consequence the workflow engine may not be notified about the completion of each executed activity.
2. To reduce costs, the workflow engine may be installed on a machine with low Service Level Agreements. Therefore the engine may occasionally fail. Suppose a particular activity – of a process under control of the workflow engine – has successfully been executed but flagging off that activity in the workflow engine fails because that engine is down at that moment. If the engine is up again, it will return that same activity since for the engine it seems not to be executed yet. The repeated execution of one activity might obviously leave the system in an incorrect state.

**A workflow engine may fail or an actor may choose not to be assisted by a workflow engine. How can we ensure that the use of a workflow engine is optional?**



**Figure 3 – Architecture for stateful process enactment**

## **Forces**

The forces can be described in terms of a set of quality characteristics that need to be considered. Furthermore it is appropriate to distinguish between forces that arise during process modeling (design time) on the one hand and process enactment (runtime) on the other hand.

- Process modeling forces
  - **Executability:** The specification of a process is interpreted by the workflow engine. Therefore the specification must be complete and correct. Activities of a process model like the one in Figure 2 are identified only by their respective names. To improve executability the workflow engine should e.g. be able to automatically determine when a particular activity can be registered as completed.
  - **Comprehensibility:** A process model must be easily understood by business people. For example, Figure 2 depicts a process model that can easily and rapidly be interpreted. However, in order to enable the automated execution of business processes (executability force), the corresponding process models should be as detailed as possible, which will most likely be at the expense of the understandability for the human actors.
  - **Flexibility:** A process model should allow the execution of every process variant, albeit without explicitly elaborating each variant. A human actor should be minimally constrained while executing activities. At the same time, an activity should only be enabled when its successful execution can be guaranteed. If multiple execution paths are possible and each path must explicitly be represented, this generally leads to complex process models.
  - **Expressivity:** A process modeling language should have a rich semantics that enables modeling all relevant requirements. The Workflow Patterns initiative (<http://www.workflowpatterns.com>) has identified different categories of patterns that should be supported by a workflow engine. A distinction is made between control-flow, data, resource, operational and exception handling patterns.
- Process enactment forces
  - **Performance:** The introduction of workflow support should not create performance issues, especially not in the existing operational information systems. As explained in the context, activity services and business object services are realized through the use of optimized mainframe transactions. During process enactment the workflow engine should influence these transactions to the minimal amount possible.
  - **Progress monitoring and enforcement:** The workflow engine must ensure that enabled activities are effectively being executed. The problem description states that an actor may execute activities without intervention of the workflow engine. However, these activities may influence the state of process instances that are maintained by the workflow engine. Therefore the workflow engine must be aware of every executed activity, so it can react appropriately.

## Solution

The only way to avoid synchronization problems between business data and process state is to avoid the explicit storage of the process state. Therefore, one should create a workflow engine that reconstructs the process state each time it is needed. Such a workflow engine is **stateless** since the conditions for enabling and completing activities are derived from the state of one or more business objects. As illustrated in Figure 4, the workflow engine reconstructs the state of a process through the use of business object services, each time the next possible activities are to be determined. As opposed to stateful process enactment, the state of a process is not explicitly stored as part of the workflow control data.

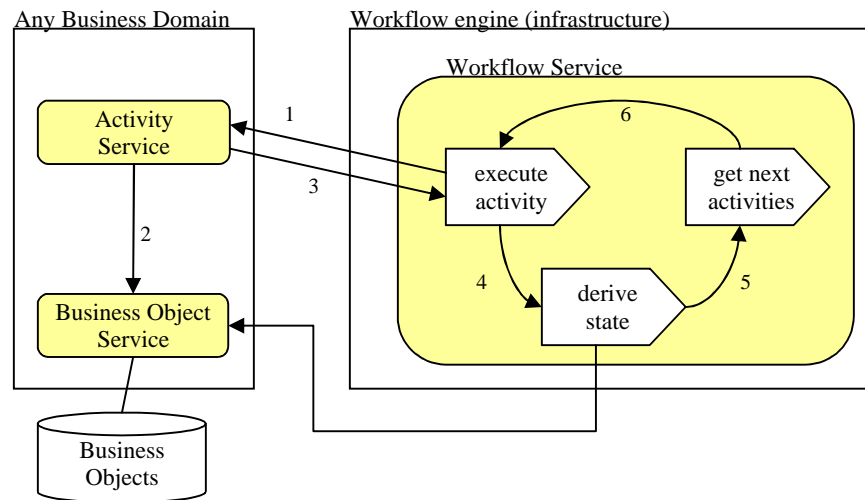


Figure 4 – Architecture for stateless process enactment

Stateless process enactment can be seen as the extension of one of the key principles of the case handling paradigm [2] that states that an activity is completed if the associated mandatory data objects are entered. As illustrated in Figure 5, both the pre-conditions and post-conditions of an activity depend on (the existence and/or state of) one or more (attributes of) business objects. If the pre-conditions of a particular activity in the context of a process are met, that activity can be executed. On the other hand, an activity is completed if all its post-conditions are met. Note that the pre- and post-conditions can be added to the activities of an existing process model. However, as we will illustrate below, a set of activities enriched with pre- and post-conditions describes an (implicit) process model on its own.

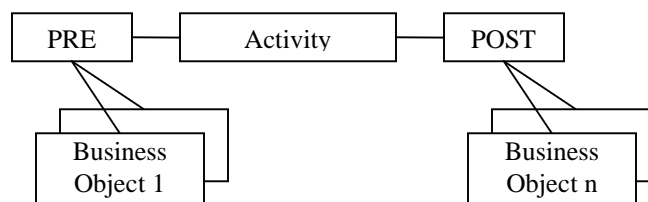


Figure 5 – Activity pre- and post-conditions

Consider the enactment of the claim handling process in a stateless way. To do so the activities must have pre- and post-conditions that are expressed in terms of business objects. An overview of the relevant business objects and their attributes is depicted in Figure 6 using the Unified Modeling Language (UML). A client has one or more contracts and each contract insures one or more objects. For the sake of simplicity, we assume that each claim case is about exactly one damaged object. A claim case can be in five different states: opened, fraudulent, genuine, paid and closed. The possible indemnity of a case can be derived from the assessed damage of an insured object.

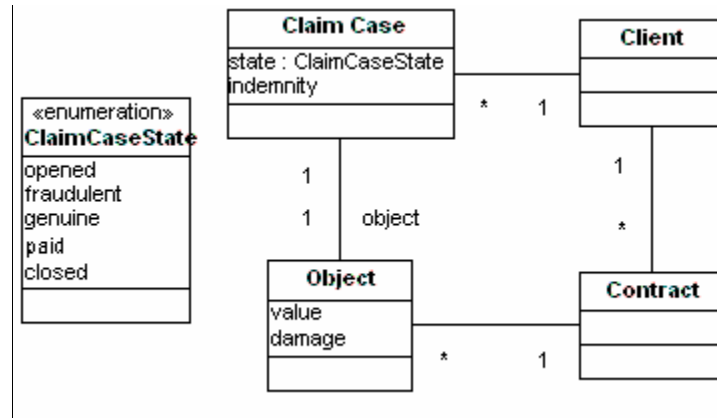


Figure 6 – Relevant business objects

Table 1 shows the set of activities and their accompanying pre- and post-conditions. These conditions can be added to the process model (Figure 2) or the set can be used as a process model on its own.

Activity	Pre-conditions	Post-conditions
Create Claim Case	/	ClaimCase cc exists AND cc.state = opened
Identify Object Damage	cc.state = opened	cc.object exists AND cc.object.damage filled out
Calculate Indemnity	cc.object.damage filled out	cc.indemnity filled out
Detect Possible Fraud	cc.object.damage filled out	IF fraud THEN cc.state = fraudulent ELSE cc.state = genuine
Pay Indemnity	cc.indemnity filled out AND cc.state = genuine	cc.state = paid
Close Claim Case	cc.state = paid OR cc.state = fraudulent	cc.state = closed

Table 1 – Claim handling activities, pre- and post-conditions

## Consequences

In most existing process modeling notations, the pre- and post-conditions on activities are often only implicitly specified as part of the control flow (arrows, messages and the conditions associated to gateway constructs). For stateless process enactment instead, one is obliged to model these conditions as they are used for deriving the process state at runtime. This clearly enhances the **executability** of the process since it is precisely defined when an activity is enabled or completed. When using typical process models, the workflow engine has no clue when an activity is completed until it receives an explicit notification. Moreover the engine can not verify the completion of the activity; it merely has to 'believe' the sender of the notification. This approach is error-prone especially for activities that require human input. On the other hand, explicitly modeled post-conditions enable the engine to verify them. Besides improving the workflow engine's executability, allowing the specification of pre- and post-conditions also extends the modelers' **expressiveness**.

The major drawback of this approach is the possibly intolerable drop in **performance** when adopting this approach. Firstly, each time the workflow engine needs to derive state, all pre- and post-conditions must be evaluated, which can be extensively time-consuming if the process consists of many activities with many conditions to be checked. A second performance issue may arise during the effective calculation of the state of an activity, especially if many attributes must be queried to do so. Assume for example that the workflow engine currently enacts a claim handling process of which it only knows the claim case ID. To reconstruct state it has to query all objects, damages, parties, etc. involved in this claim case to collect their attributes.

The second performance issue during state reconstruction can be alleviated by increasing the use of "meta-state" attributes of business objects. Those attributes explicitly indicate the state of the object. For example, the state of an order (requested, shipped, paid, etc.) is often added as an attribute of order, or the claim case state in Figure 6 is a meta-state attribute of a claim case. Although such attributes facilitate the reconstruction of process state, one should not be tempted to add attributes to business objects only for the sake of a particular process. This obviously moves process information in the business objects and consequently impedes process flexibility. Note however that the addition of attributes to business objects does not reintroduce the problems of the stateful approach since all state information is kept in the business objects.

As indicated in the solution, it is possible to add pre- and post-conditions to an existing process model or it is possible to specify a set of activities *only* in terms of its pre- and post-conditions. This way a process can be seen as an unstructured set of activities in which an activity can be executed as soon as its pre-conditions are met. Such a specification offers the most **flexibility** since appropriate execution paths can be inferred at runtime. For example, according to the process model in Figure 2, it is possible to execute activity "Detect Possible Fraud" only after "Identify Object Damage" is executed, while according to Table 1 this is allowed as soon as the claim case is created. This way the claim handler has more freedom in deciding when to execute the activity.

As another example, consider a claim handler who finds evidence to have the activity “Detect Possible Fraud” executed again at some arbitrary point in time after its first execution. Obviously, the process model in Figure 2 must be heavily extended to support this plausible scenario, whereas in the stateless approach, the claim handler can schedule this activity as long as its pre-condition is satisfied.

In the stateful approach process progress is monitored and enforced in the workflow engine by updating the state of the process after a completed activity returns control to the workflow engine. If particular activities become enabled, the engine has three options: (a) the engine waits for an actor to execute the activity; (b) the engine executes the activity if it requires no actor input; (c) the engine sends one or more tasks to actors who should execute the activity. The workflow engine can always send notifications when activity deadlines have to be respected.

In the stateless approach it is more complex to **monitor and enforce progress** if an activity that requires actor input is executed without assistance of the workflow engine. Indeed, in this case the workflow engine has no clue when that activity is completed. Despite this fact there are two possible approaches to make sure that progress is enforced. Firstly, the workflow engine can regularly calculate the state of a process to see if activities must be executed or tasks must be distributed. Given the previously discussed performance issues, it is indispensable to set an appropriate polling frequency. A second approach avoids this thorny task, namely by using a publish/subscribe mechanism: the workflow engine subscribes to particular data changes (published by business object services) that influence activity pre- and post-conditions, so the workflow engine reacts only when necessary. A separate component in the workflow engine should (a) manage the subscriptions for each process and (b) possibly group multiple notifications before state is recalculated.

A control-flow based notation requires the explicit modeling of each process variant, which must be represented by a path in the model. For complex processes, this leads to gigantic models that are difficult to **comprehend**. On the contrary, processes with only a few variants can easily be represented and interpreted using graphical flow chart-like notations like BPMN. In addition, many tools offer animation to visualize and simulate process execution, which makes them indispensable for business people that need to assess as-is as well as to-be processes in a quick and efficient way. Finally, it is very hard to understand and verify the eventual behavior of a group of activities that are only specified by means of pre- and post-conditions. Even small process models, such as the one represented in Table 1, are quite complex to understand.

### ***Known uses***

As discussed in [2], only a few workflow management systems support the case handling paradigm, such as FLOWer [3] and COSA [4], which implements part of this pattern: (a) only post-conditions of an activity can be expressed in terms of mandatory data objects that must exist and (b) a data object is the single fine-grained construct to express post-

conditions. The open-source workflow engine, called con:cern (<http://con-cern.org>), fully applies this pattern.

## ***Acknowledgements***

The authors would like to express their gratitude to our shepherd António Rito Silva, whose suggestions greatly enhanced this pattern. This pattern has been written as part of the KBC-Vlekhov-K.U.Leuven research chair on 'Service and Component Based Development' sponsored by KBC Banking & Insurance.

## ***References***

- [1] WFMC: The workflow reference model. Technical Report WFMC-TC-1003, Workflow Management Coalition (1995)
- [2] van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. *Data and Knowledge Engineering* 53(2) (2005) 129–162
- [3] Pallas Athena: Case Handling with FLOWer: Beyond workflow. Pallas Athena BV, Apeldoorn, The Netherlands (2002)
- [4] Software-Ley: COSA Activity Manager. Software-Ley GmbH, Pullheim, Germany (2002)