Ardagna, D., Baresi, L., Comai, S., Comuzzi, M. & Pernici, B. (2011). A Service-Based Framework for Flexible Business Processes. IEEE Software, 28(2), pp. 61-67. doi: 10.1109/MS.2011.28

A Service-based Framework for Flexible Business Processes

Danilo Ardagna[1], Luciano Baresi[1], Sara Comai[1],
Marco Comuzzi[2], Barbara Pernici[1]

[1]Politecnico di Milano, Italy

[2] Eindhoven University of Technology, the Netherlands

Abstract

This article describes a framework for the design and enactment of flexible and adaptive business processes. It combines design-time and run-time mechanisms to offer a single integrated solution. The design-time environment supports the specification of process-driven Web applications with Quality of Service (QoS) constraints and monitoring annotations. The run-time identifies the actual services, from the QoS perspective, oversees the execution through monitoring, and reacts to failures and infringement of QoS constraints. The article also discusses these issues on a proof of concept application developed for an industrial supply chain scenario.

Keywords: Business process, QoS, Web design, Web service, adaptability.

1 Introduction

Globalization and accessibility are imposing a significant shift in business processes. Static solutions are leaving the stage to flexible processes conceived to address rapidly changing business needs. For example, the Internet as a platform is fostering the idea of *virtual supply chains*, where business partners change seamlessly as soon as new business opportunities arise.

The market requires business processes that can be sliced into self-contained elements and be re-composed on demand. The dynamism and flexibility of these business models impose that supporting information systems change and evolve as fast as the business they support. Most of the changes cannot be applied during conventional maintenance, but must be performed transparently on running systems.

These new requirements are making information systems evolve from closed and proprietary entities to open solutions based on non-proprietary standards [9]. The service-oriented paradigm is providing the enabling infrastructure for this shift and complex information systems are becoming suitable compositions of services —often referred to as processes. Services can be performed by humans and integrated in the process through dedicated user interfaces, or can be automated and provided as Web services.

In a continuously evolving context, where in many cases the actual services can only be selected at run-time, flexible and adaptive solutions are required to guarantee both the functional requirements and quality of service (QoS) of the process. Flexible business processes also impose a shift in validation: Conventional pre-deployment testing is not enough anymore since it cannot foresee run-time changes; self-validation and self-adaptation techniques must be introduced, to monitor services and apply corrective actions as soon as problems arise.

These are the underpinnings of the framework DISCoRSO[1] (Distributed Information Systems for COoRdinated Service Oriented interoperability), which offers a comprehensive service-based solution for the specification and management of flexible and responsive business processes. At design level, it proposes an "annotated" BPMN (Business Process Modeling Notation) to specify the entire process including both manual activities and automated services. Human-based activities are modeled as Web pages through WebML [4]. Automated activities are modeled as "abstract" interactions with external Web services and specified in terms of functional requirements and QoS constraints. Both classes of activities can be annotated with *supervision rules* to probe their execution at run-time and trigger corrective actions if needed.

Before executing a process, the framework replaces abstract services with "concrete" ones by selecting those services that best satisfy the functional requirements and fit the QoS constraints specified through annotations. At run-time, it oversees the execution of the different activities and reacts accordingly in case of failures or QoS violations.

[1] www.discorso.eng.it

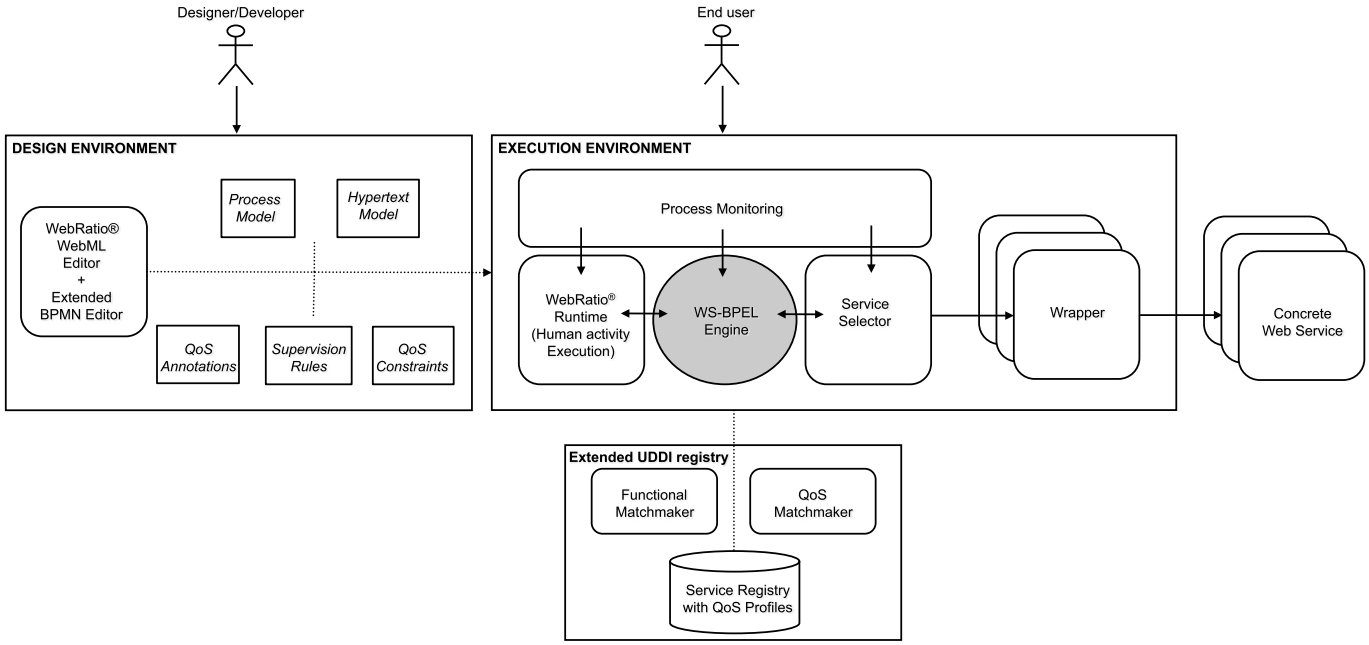

Figure 1: The Architecture of the DISCoRSO Framework

This article introduces the framework (Section 2) and exemplifies it on the development of a proof of concept cooperative business process borrowed from the textile supply chain (Section 3) studied in the DISCoRSO project. It also briefly discusses related approaches (Section 4) and draws some conclusions (Section 5).

2 DISCoRSO Framework

Figure 1 shows the main components of the framework. At design-time, The CASE tool WebRatio[2] provides an extended BPMN editor to allow designers to define business processes with both activities carried out by humans and automated activities performed by Web services. In the former case, the framework supports the development of the Web interface of each activity, through the WebRatio WebML editor. In the latter case, it requires that *abstract* services be specified in terms of required operations. Complete processes, fragments (i.e., set of activities), and single

[2]www.webratio.com

service invocations are annotated with QoS constraints (e.g., response time, cost, reputation, and reliability) and statistics, like the probability of executing branches and loops. These annotations are then used at run-time to guide the selection of the actual services. All the activities can also be augmented with supervision rules to oversee their execution and trigger corrective actions if needed.

Starting from the extended BPMN specification, the CASE tool automatically derives a skeleton of a WebML [4] hypertext extended with primitives controlling the execution of the whole process (e.g., activities start and the end), specifying the containers for the manual activities and the hooks for the automated ones. The containers of the manual activities are then refined with the specification of their content and navigational paths, thus defining the Web interfaces that will be shown to the users. The extended hypertext can be automatically transformed into application code executable on top of the Java2EE, Struts, and .NET platforms. The tool also translates the fragments of the BPMN process that correspond to automated activities into BPEL (Business Process Execution Language) processes for their execution. These processes only comprise abstract services; the actual selection will occur at run-time.

Concrete Web services (services supported by real implementations) are stored in an extended UDDI registry, where they are associated with QoS profiles. This registry also embodies a functional and a QoS matchmaker [1]. Functional matchmaking is driven by a similarity algorithm based on services' descriptions, whereas the QoS matchmaker filters candidate Web services according to the QoS constraints associated with the abstract placeholders. The *Service Selector* uses these features to find the services that implement the abstract services and best fit the constraints.

At run-time, the *WebRatio run-time* orchestrates the whole process, by supporting the execution of the manual activities and interacting with the *BPEL engine* to invoke retrieved Web services. The invocation of concrete Web services is obtained by implementing a late binding mechanism through wrappers [1]. Supervision rules are used by the *Process Monitoring* to assess process executions and trigger adaptation mechanisms automatically as soon as anomalies are detected. If

a concrete Web service is faulty or violates a constraint, the framework uses the *Service Selector* to find a substitute satisfying the QoS constraints in the current state of the process.

2.1 Run-time Service Selection

Abstract service specifications are mapped onto concrete services by exploiting the process annotations defined by the designer. Service selection is modeled as an optimization problem performed when the business process is instantiated and also iterated at run-time to take into account performance variability and invocation failures (notified by the *Process Monitoring* module presented in the next section).

The optimization goal is to maximize the average QoS perceived by the user and consider both *local* and *global* constraints. Local constraints can predicate on the properties of a single activity (or abstract service), while global constraints specify requirements on a set of activities or at process level.

The average QoS is evaluated statistically from the probability of executing conditional branches and the distribution of the number of iterations in loops. Statistics can be either estimated at design-time by the designer or updated at run-time by the monitoring component from past executions [3]. The satisfaction of global constraints is known to be a NP-hard problem. A number of solutions have been proposed to reduce this complexity, guaranteeing global constraints only for the critical path (i.e., the path having the highest execution time) or statistically. Our framework exploits a new optimization approach [2] based on mixed integer linear programming models, which overcomes the limitations of the previous solutions, supports the selection of stateful Web services, and is particularly effective under severe QoS constraints.

Furthermore, if a feasible solution of the optimization problem does not exist, the approach negotiates the QoS parameters with the service providers to find a sub-optimal solution and thus reduce the failure rate [2]. To this end, we implemented bilateral iterated techniques [6] and we also exploited a novel approach based on lightweight offer configuration strategies [5], where the

service providers express QoS profiles in terms of discrete values (e.g., response time may be 2, 4, or 6 days). Providers also specify a pricing model associated with each QoS dimension, while the designer specifies a negotiation strategy to distribute the extra-budget among the different QoS dimensions. At run-time, re-optimization is activated whenever a QoS constraint is violated. It leads to either the selection of new candidates for the execution of the remaining activities or maintains the already selected candidates, but with a different (re-negotiated) QoS profile.

2.2 Supervision

Supervision rules comprise both monitoring assertions and recovery actions. Monitoring assertions take the form of pre- and post-conditions and predicate on both functional correctness and QoS constraints. These expressions are specified in a variant of WSCoL (Web Service Constraint Language, [3]). WSCoL provides language-specific constructs for *data collection* and *data analysis*.

Data collection is responsible for obtaining the information used to check whether the activities match the specified set of constraints and to update the execution statistics on branches and loops (see Section 2.1). The language distinguishes among three kinds of monitoring data: Data belonging to the state of the running process, data obtained externally from any remote component that exposes a WSDL interface, and data obtained from previous process executions (since collected data can be made persistent).

Data analysis checks whether collected data comply with set requirements and highlights possible variations. It supports the typical boolean, relational, and mathematical operators, predicates on sets of values through the use of universal and existential quantifiers, and also supplies other useful constructs (e.g., the max, min, and average values of collected data).

Recovery actions can trigger adaptation mechanisms provided by the run-time infrastructure, invoke particular external applications implemented as Web services, or notify the violations to the system administrator and maybe to the user. For example, if a concrete Web service started behav-

ing incorrectly or providing responses lately, the system would automatically notify the problem to the administrator and temporarily remove the service from the registry to avoid further selections by other processes.

Each rule also comprises a priority, used to tailor the amount of supervision actually performed at run-time. This value is compared to the priority with which the whole process is executed to decide whether the rule has to be evaluated: If the value is greater than or equal to the global priority, the rule is executed, otherwise it is skipped. A process console allows the user to set the global execution priority, and also visualizes the current state of the different running instances, along with violated constraints and the values that caused such violations.

3 Case study

The effectiveness of the framework has been assessed through a proof of concept business process borrowed from the silk-textile district in Northern Italy, one of the districts studied by the DiSCoRSO project. The order fulfilment process reflects a typical scenario where different small and medium enterprises —in the same geographical area— can perform the same activity with different quality levels (e.g., in terms of response time, availability, reputation, and price).

Figure 2 presents the main activities of the BPMN process. The converter plays a focal role: from a business perspective, it acts as an intermediary between the corporate customers, that is, clothing or fashion companies, and the partners (sub-contractors) in the district; from a technological standpoint, it is the orchestrator of the whole process. Sub-contractors expose their (legacy) services as Web services, and publish them onto the extended UDDI registry.

The converter is supposed to select the best services that can perform required activities according to QoS constraints. In the current practice, this selection is performed through complex and lengthy interactions between company managers, leading to long-term contracts that cannot be easily modified. The example —and the whole project— demonstrates the feasibility of more automated and flexible solutions to pave the ground to optimized and responsive processes and

micro-contracts. To this end, the framework automatically selects the best services at run-time, monitors the fulfillment of QoS guarantees, and undertakes corrective actions as soon as problems arise.

As soon as the converter receives an order, it checks whether the requested items are available in stock. If not, it prepares a dispatch request, which is sent to a weaving mill for the first processing stage. After preparing the tissue, the weaving mill forwards the order to a dyer to complete the processing. Then, the quality control checks and certifies the quality of the final product. Quality checks are classified as *basic* (if they only require visual analyses to identify defects or holes) or *advanced* (if they require chemical analyses to determine fiber properties, e.g., color drying). If any item included in the order does not pass the first quality check, a second control must be performed. Real historical data provided by the district allowed us to estimate that this only happens in the 10% of the cases (probability $p_2 = 0.1$ in Figure 2).

The activities performed by the converter, the weaving mill, and the dyer are executed by humans and are supported by Web interfaces. In the left top corner of Figure 2 a snapshot of the WebML model of activity converter order processing is presented: it contains a page to show the data of the received order and enable the selection of the items needed to satisfy the request.

Quality control activities are performed by automated services, represented by means of a BPEL specification, automatically obtained from the BPMN process.

The BPMN editor supports the specification of properties (Figure 2 in the left bottom corner), where QoS constraints, process statistics, and monitoring rules can be associated with the whole process or with specific activities. In our application we added the following global constraints: (i) the two quality control activities have to be performed by two different providers, (ii) the overall process should take less than two weeks, and (iii) the cost of the whole process should be less than 10K euros. Furthermore we have introduced a local constraint on order processing activities reputation which has to be greater than 0.9. Service reputation is defined as the percentage of service invocation such that the service execution time does not exceed the maximum execution

time published as threshold in the registry (see Figure 3). When a new service is added to the Extended UDDI registry, its reputation value is always set to 1, that is, to the maximum value.

At run-time, local and global constraints are monitored and in case of failures or QoS violations a re-optimization of the process is triggered. An example of rule for monitoring constraint can be specified as follows: Each time an activity terminates, we retrieve its execution time; if it is greater than the one originally associated with the activity, and we foresee that the global execution time of the process may exceed the two weeks, we start a new selection (optimization) to identify faster services, with acceptable QoS, to complete the execution:

```
if ($execTime >= retrieve($expExecTime, $actId)) &&
     call(exptedTime($procId)) > 14d
then notify($emailAddr) && call(serviceSelector($procId))
```

The rule uses the special-purpose variable `$execTime`, which stores the execution time of the last completed activity, and retrieves the expected one (`$expExecTime`), which is the threshold stored in the registry, by using the activity's id `$actId`. It also calls an external functionality, `exceptedTime`, to compute the foreseen execution time for the whole process, identified through its id `$procId`. If the actual time is greater than the planned one and the predicted execution time for the whole process is likely to exceed the two weeks (14 days), then it notifies the system administrator, whose email address is `$emailAddr` and invokes the service selector to starts a new optimization process. Constraints (ii) and (iii) can be specified likewise.

Supervision is also used to update the statistics about execution. In particular the following rule is associated with the *SW-quality check* switch of Figure 3:

```
store("SW-quality check", $condValue)
```

The rule stores the different values (`condValue`) with which the switch is executed. Since accepted values are only *true* and *false*, we can easily compute the statistics about the element by

dividing the number of positive (*true*) outcomes by the total number of executions, which is the sum of both *true* and *false* ones.

The framework can also deal with incorrect or faulty behaviors by setting proper supervision rules. For example, the following rule is used to count the number of consecutive failures of the service identified by `$srvId`. If this number is greater than 3, then the Web service is removed from the registry and an email is sent to the system administrator. The approximation used here is that a service is faulty if its answer does not come within a given timeframe (equal to `VALUE` in the rule). Note that this covers also the case of answers never received.

```
let $execTimes = retrieve($execTime, $srvId, 10)
if (count($execTimes, $time > VALUE) > 3)
then call(remove($srvId)) && notify($emailAddr)
```

Figure 3 shows an example of the run-time execution of the quality control activities. Note that concrete services are selected dynamically according to QoS constraints and to the requested kind of quality checks. In the example, we assume that the current order needs *advanced* quality checks. We also consider that the warehouse does not have the items required by the customer in stock and hence the system must place an order to the weaving mill and dyer. Their activities have been already completed, in 10 days and a cost of 6K euros, and we are ready for the quality control activities.

The table in Figure 3 lists the available concrete services: service *C* is able to perform only basic analyses and therefore is not considered by the Service Selector. Considering the constraints, service *B* is discarded due to a reputation value lower than 0.9. Also service *E* does not lead to admissible solutions, since both *E+A* or *E+D* quality checks require 5 days. The Service Selector can then only select services *D* and *A*, with an expected total cost of 9K euros and execution time equal to 14 days (computed by summing the costs/execution time of the selected services to the cost/execution time of the previous phases), thus satisfying the global constraints. Now we assume that service *D* takes 3 days instead of 2 to complete: A supervision rule detects the delay on its

execution time and calls the run-time optimization again; in this case a negotiation is triggered and the second quality check is performed by service *A* in one day but with a cost of 3K euros. The 1K euro initial extra-budget is exploited in the negotiation process to reduce the execution time of the second quality control and to meet the 14-day global constraint.

4 Related work

To the best of our knowledge, we are not aware of any other proposal that integrates the design of process models with both human-based and automated activities, the QoS-based dynamic selection of services, and their run-time supervision.

As far as the *design of business processes and their Web interfaces* are concerned, different Web Engineering methodologies (e.g., OO-H and UWA [8]) have extended their notations conceived for the design of Web applications with business process primitives, but never addressing the integration with QoS-based selection/optimization/supervision. Web-enabled workflow management systems, such as, for instance, IBM WebSphere and Oracle Workflow, also provide integration of processes and hypertexts. These tools, however, solely use the Web as a (thin) interface to proprietary software modules.

From the *QoS* perspective, the need for extending traditional service-based platforms by also considering quality aspects is an emerging research challenge [9]. Our approach for the execution of flexible applications draws from the PAWS framework [1], which provides advanced mechanisms for service retrieval, selection, and mediation. With respect to PAWS, the DISCoRSO approach provides a richer framework, which couples QoS-based service selection with the automated generation of user interfaces and run-time supervision.

As for monitoring and supervision, several proposals define specification languages for Service Level Agreements (SLAs) and propose an associated monitoring architecture [10, 11]. These approaches, however, focus on high-level contracts, but they do not scale down to individual process' constructs, which is one of the key elements of our approach. Quite differently, Jurca et

al. [7] propose an approach for QoS monitoring based on quality ratings from the clients. Our approach stresses the synchronous intertwining between execution and supervision to be able to detect anomalies as soon as they arise.

5 Conclusions and Future Work

The article introduces the elements of the framework DISCoRSO for the design and enactment of flexible and dynamic business processes. The key innovation of the framework lays in the integrated design of complex processes involving both human and automated activities, with run-time QoS-based service selection and supervision. This paves the ground to flexible and dynamic solutions, where self-validation and self-adaptation techniques can be introduced. A proof of concept application has also been discussed: The adoption of the proposed approach allows one to better coordinate the activities within a cluster of companies and to choose the best "aggregation" of services (offered by the companies in the district) to reach a given goal, and modify such "aggregations" at run-time according to QoS constraints.

From a research standpoint, we are planning to extend service selection by reducing the optimization overhead to facilitate the management of multiple instances of the same business process and to investigate constraints on the mutual correctness of different processes.

Authors' biographies

- **Danilo Ardagna** is an assistant professor at Politecnico di Milano. His research interests include services composition and autonomic computing.
- **Luciano Baresi** is an associate professor at Politecnico di Milano. His research interests concern the design, deployment, and supervision of fully distributed, dynamic, and dependable software systems.
- **Sara Comai** is an associate professor at Politecnico di Milano. Her main research interests include conceptual modeling of complex data-intensive workflow and service based Web applications.

- **Marco Comuzzi** is an assistant professor at Eindhoven University of Technology. His research interests concern automated negotiation algorithms applied in the context of Web service quality.

- **Barbara Pernici** is full professor of Computer Engineering at Politecnico di Milano. Her research activity is in the area of information and service engineering. She has been chair of IFIP WG 8.1 on Information Systems Design and of IFIP Technical Committee TC8 Information Systems.

References

[1] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani. PAWS: A famework for executing adaptive Web-service processes. *IEEE Software*, 24(6):39–46, 2007.

[2] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Trans. on Software Engineering*, 33(6):369–384, June 2007.

[3] L. Baresi and S. Guinea. Towards dynamic monitoring of WS-BPEL processes. In *ICSOC*, pages 269–282, 2005.

[4] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann Publishers Inc., 2002.

[5] M. Comuzzi and B. Pernici. A framework for QoS-based Web service contracting. In *ACM Transactions on the Web*, volume 3, pages 1–52, 2009.

[6] P. Faratin, C. Sierra, and N. R. Jennings. Negotiation decision functions for autonomous agents. *Int. Journal of Robotics and Autonomous Systems*, 23(3-4):159–182, 1998.

[7] R. Jurca, W. Binder, and B. Faltings. Reliable QoS monitoring based on client feedback. In *Proc. WWW2007*, pages 1003–1012, 2007.

[8] N. Koch, A. Kraus, C. Cachero, and S. Meli. Integration of business processes in web application models. *Journal of Web Engineering*, 3(1):22–49, 2004.

[9] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: a Research Roadmap. *Int. J. Cooperative Inf. Syst*, 17(2):223–255, 2008.

[10] A. Sahai, V. Machiraju, M. Sayal, L. J. Jin, and F. Casati. Automated SLA monitoring for web services. In *Proc. IFIP/IEEE International Workshop on Distributed Systems*, pages 28–41, 2002.

[11] J. Skene, A. Skene, J. Crampton, and W. Emmerich. The monitorability of service-level agreements for application-service provision. In *WOSP 2007 Proc.*, pages 3–14, New York, NY, USA, 2007. ACM Press.

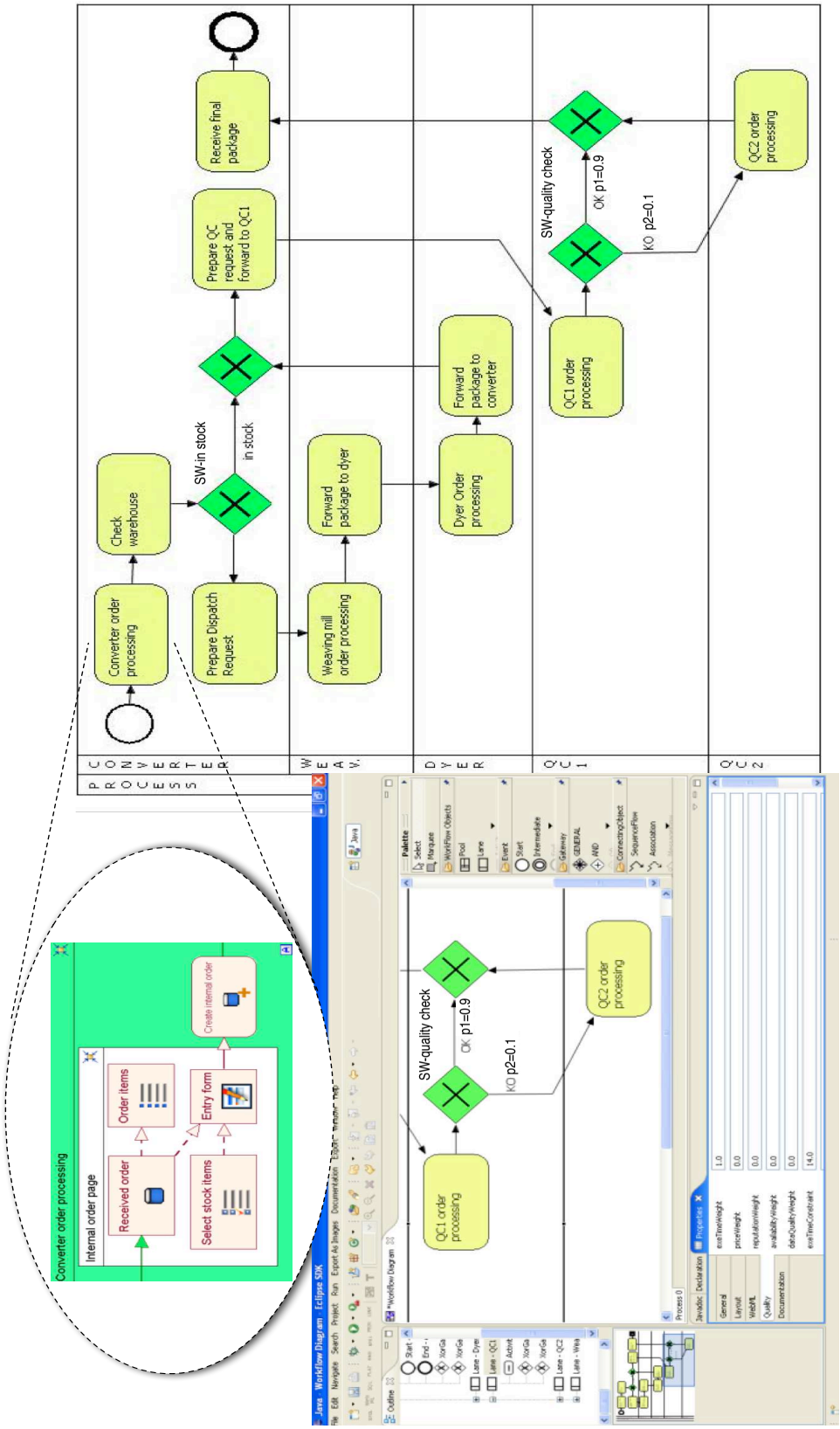

Figure 2: BPMN model of the example process.

Figure 3: Execution of the quality control activities.