GLOBALIZATION AND ACCESSIBILITY have caused a significant shift in business processes, from static solutions to flexible processes that can address rapidly changing business needs. Consider virtual supply chains, where business partners change seamlessly as new business opportunities arise. The marketplace needs business processes that can break into independent elements and then recompose on demand. Dynamic, flexible business models require supporting information systems that evolve as quickly as the businesses they support. Most of the changes can’t be applied during conventional maintenance but must be performed transparently on running systems.

These new requirements are forcing information systems to evolve from closed, proprietary entities to open solutions based on public standards. The service-oriented paradigm provides the enabling infrastructure for this shift, and complex information systems are becoming suitable compositions of services—often called processes. Humans can perform services and integrate them in the process through dedicated user interfaces; services can also be automated and provided as Web services.

In a continuously evolving context, where in many cases you can select the actual services only at runtime, flexible and adaptive solutions are required to guarantee both the process’s functional requirements and quality of service (QoS). Flexible business processes also impose a validation shift: conventional predeployment testing isn’t enough anymore because it can’t foresee runtime changes; we must introduce self-validation and self-adaptation techniques to monitor services and apply corrective actions as soon as problems arise.

These are the underpinnings of the Discorso (Distributed Information Systems for Coordinated Service-Oriented Interoperability) framework (www.discorso.eng.it), which offers a comprehensive service-based solution for specifying and managing flexible and responsive business processes. (For a discussion of other approaches, see the “Related Work in Business Process Design” sidebar.) At the design level, Discorso proposes an “annotated” business process modeling notation (BPMN) to specify the entire process, including both manual activities and automated services. It models human-based activities as webpages through WebML and automated activities as “abstract” interactions, with external Web services specified in terms of functional requirements and QoS constraints. We can annotate both classes of activities with supervision rules to probe their execution at runtime and trigger corrective actions if needed. Then, 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
RELATED WORK
IN BUSINESS PROCESS DESIGN

We’re not aware of any other proposal that integrates process model design with both human-based and automated activities, the QoS-based dynamic selection of services, and their runtime supervision.

Regarding the design of business processes and their Web interfaces, different Web engineering methodologies—such as Object Oriented Hypermedia (OO-H) and Ubiquitous Web Applications (UWA)—have extended their notations for Web application design with business process primitives, but they haven’t addressed integration with QoS-based selection, optimization, and supervision. Web-enabled workflow management systems, such as IBM WebSphere and Oracle Workflow, also integrate processes and hypertexts. These tools, however, use the Web solely 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. Our approach for executing flexible applications draws from the PAWS framework, 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 runtime supervision.

As for monitoring and supervision, several proposals define specification languages for service-level agreements and propose an associated monitoring architecture. These approaches focus on high-level contracts, but they don’t scale down to individual processes’ constructs, which is a key element of our approach. Quite differently, Radu Jurca and colleagues propose an approach for QoS monitoring based on quality ratings from the clients. Our approach stresses the synchronous intertwining between execution and supervision to detect anomalies as soon as they arise.

References

Discorso Framework

Figure 1 shows the framework’s main components. At design-time, the CASE tool WebRatio (www.webratio.com) provides an extended BPMN editor to let designers define business processes with both human and automated activities. In the former case, the framework supports the development of each activity’s Web interface, through the WebRatio WebML editor. In the latter case, abstract services must be specified in terms of required operations. Complete processes, fragments (that is, sets of activities), and single-service invocations are annotated with QoS constraints (such as response time, cost, reputation, and reliability) and statistics (such as the probability of executing branches and loops). The framework then uses these annotations at runtime to guide the selection of the actual services. You can also augment the activities 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 hypertext extended with primitives controlling the entire process’s execution, specifying containers for manual activities and hooks for automated ones. The designer then refines the manual activity containers with the specification of their content and navigational paths, thus defining the Web interfaces that users will see. We can automatically transform the extended hypertext 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 runtime.

An extended UDDI registry stores concrete Web services (those supported by real implementations) and associates them with QoS profiles. This registry also embodies a functional and a QoS matchmaker. Driving functional matchmaking is a similarity algorithm based on service 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 runtime, WebRatio orchestrates the whole process by supporting the execution of 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. Process monitoring uses supervision rules 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.

**Runtime Service Selection**

The framework maps abstract service specifications onto concrete services by exploiting designer-defined process annotations. Service selection is an optimization problem performed when the business process is instantiated and also iterated at runtime to account for performance variability and invocation failures.

The optimization goal is to maximize the user’s average QoS 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 the 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 runtime by past executions’ monitoring components. Satisfying global constraints is a known NP-hard problem. Researchers have proposed several solutions to reduce this complexity, guaranteeing global constraints only for the critical path (that is, the path with the highest execution time) or statistically. Our framework exploits a new optimization approach based on mixed-integer linear programming models: it overcomes the previous solutions’ limitations, supports the selection of stateful Web services, and is particularly effective under severe QoS constraints.

Furthermore, if a feasible solution of the optimization problem doesn’t exist, the framework negotiates QoS parameters with service providers to find a suboptimal solution and thus reduce the failure rate. To this end, we implemented bilateral iterated techniques and also exploited a novel approach based on offer configuration strategies, in which service providers express QoS profiles in terms of discrete values (for example, 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 runtime, reoptimization occurs whenever a QoS constraint is violated, resulting in either the selection of new candidates to execute the remaining activities or the maintenance of the already selected candidates but with a different (renegotiated) QoS profile.

**Supervision**

Supervision rules comprise both moni-
Monitoring assertions take the form of pre- and postconditions and predicate on functional correctness and QoS constraints. These expressions are specified in a variant of WSCoL4 (Web Service Constraint Language), which provides language-specific constructs for data collection and data analysis.

During data collection, the framework obtains the information used to check whether the activities match the specified set of constraints and to update execution statistics on branches and loops. The language distinguishes among three kinds of monitoring data: that belonging to the state of the running process, that obtained externally from any remote component exposing a WSDL interface, and that 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 (for example, the max, min, and average values of collected data).

Recovery actions can trigger adaptation mechanisms provided by the runtime infrastructure, invoke particular external applications implemented as Web services, or notify the system administrator and maybe a user about violations. For example, if a concrete Web service started behaving incorrectly or providing late responses, the system would automatically notify the administrator and temporarily remove the service from the registry to avoid further selections by other processes.

Each rule also contains a priority, used to tailor the amount of supervision actually performed at runtime. The system compares this value to the priority with which the whole process is executed to decide whether the rule must be evaluated: if the value is greater than or equal to the global priority, the rule is executed; otherwise, it’s skipped. A process console lets users set the global execution priority and visualizes the current state of the different running instances, along with violated constraints and the values that caused such violations.

**Case Study**

We assessed the framework’s effectiveness through a proof-of-concept business process borrowed from the silk-textile district in Northern Italy (one of the districts the Discorso project studied). The order fulfillment process reflects a typical scenario in which different small- and medium-sized enterprises—in the same geographical area—can perform the same activity with different quality levels (for example, in terms of response time, availability, reputation, and price).

Figure 2 presents the BPMN process’s main activities. The converter plays a focal role. From a business perspective, it intermediates between corporate customers—that is, clothing or fashion companies—and partners (subcontractors) in the district. From a technological standpoint, it orchestrates the whole process. Subcontractors expose their (legacy) services as Web services and publish them onto the extended UDDI registry.
The converter selects the best services that can perform required activities according to QoS constraints. Currently, this selection takes place through complex, lengthy interactions between company managers, leading to long-term contracts that aren’t easily modified. Our example—and the entire project—demonstrates the feasibility of more automated and flexible solutions to pave the way for optimized and responsive processes and microcontracts. To this end, the framework automatically selects the best services at runtime, 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 and sends it 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, quality control checks and certify the final product’s quality. Quality checks are considered basic (if they only require visual analyses to identify defects or holes) or advanced (if they require chemical analyses to determine fiber properties, such as, for example, color drying). If any item included in the order doesn’t pass the first quality check, the system must perform a second. We used real historical data from the district to estimate that this only happens in 10 percent of cases (in Figure 2, probability \( p_2 = 0:1 \)).

Humans execute and Web interfaces support the activities performed by the converter, the weaving mill, and the dyer. The top left corner of Figure 2 shows a snapshot of the webpage’s WebML model, which supports the order processing activity. It contains a page to show data about the received order and enable the selection of the items needed to satisfy the request.

Automated services perform quality control activities, represented as a BPEL specification automatically obtained from the BPMN process.

The BPMN editor supports the specification of properties (see the bottom left corner of Figure 2), where we can associate QoS constraints, process statistics, and monitoring rules with the whole process or with specific activities. We added three global constraints to our application:

1. The two quality control activities must be performed by two different providers.
2. The overall process should take less than two weeks.
3. The whole process should cost less than 10,000 euros.

Furthermore, we introduced a local constraint on order processing activities reputation, which must be greater than 0.9. Service reputation is defined as the fraction of service invocation providing an execution time lower than the maximum execution time advertised by service providers and published as threshold in the UDDI 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 runtime, the system monitors local and global constraints and, in case of failures or QoS violations, triggers a reoptimization of the process. We can specify an example of a rule for monitoring constraint as follows: each time an activity terminates, we retrieve its execution time. If it’s greater than the one originally associated with the activity, and we foresee that the process’s global execution time could exceed the two weeks, we start a new selection (optimization) to identify faster services, with acceptable QoS, to complete the execution:

\[
\begin{align*}
\text{if } & (\text{SexeTime} \geq \text{retrieved(SexpExecTime, SactId)}) \& \& \\
& \text{call(expTime(SprocId))) > 14d}
\end{align*}
\]

The rule uses the special-purpose variable \( \text{SexeTime} \), which stores the last completed activity’s execution time, and retrieves the expected execution time \( \text{SexpExecTime} \), which is the threshold stored in the registry, by using the activity’s id \( \text{SactId} \). It also calls an external functionality, \( \text{expTime} \), to compute the foreseen execution time for the whole process, identified through its ID \( \text{SprocId} \). 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), the system notifies the system administrator, whose email address is \( \text{SemailAddr} \), and invokes the service selector to start a new optimization process. Constraints 2 and 3 can be specified likewise.

We also use supervision to update the statistics about execution. In particular, the following rule is associated with Figure 3’s SW-quality check switch:

\[
\text{store("SW-quality check", ScondValue)}
\]

The rule stores the different values \( \text{ScondValue} \) with which the switch executes. Because accepted values are only true and false, we can easily compute the element’s statistics 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, we use the following rule to count the number of consecutive failures of the service identified by \( \text{SsvId} \). If this number is greater than three, then the system removes the Web service from the registry and sends an email to the system administrator. The approximation
we use here is that a service is faulty if its answer doesn’t come within a given time frame (equal to \textbf{VALUE} in the rule). Note that this also covers the case of answers never received.

\begin{verbatim}
let SexecTimes =
retrieve(SexecTime, SsrvId, 10)
if (count(SexecTimes, Stime > VALUE) > 3)
then call(remove(SsrvId))
& & notify(SemailAddr)
\end{verbatim}

Figure 3 shows an example of the runtime execution of the quality control activities. The system dynamically selects concrete services 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 doesn’t have the items the customer needs in stock and hence the system must place an order to the weaving mill and dyer. These activities have already been completed, in 10 days and at a cost of 6,000 euros, so we’re ready for the quality control activities.

The table in Figure 3 lists the available concrete services: service C can perform only basic analyses, therefore the service selector doesn’t consider it. Considering the constraints, service B is discarded due to a reputation value lower than 0.9. Also, service E doesn’t lead to admissible solutions, since both E + A and E + D quality checks require five days. The service selector can then only select services D and A, with an expected total cost of 9,000 euros and execution time equal to 14 days (computed by adding the costs and execution time of the selected services to those of the previous phases), thus satisfying the global constraints. Now, we assume that service D takes three days instead of two to complete: a supervision rule detects the delay on its execution time and calls the runtime optimization again; in this case, a negotiation is triggered, and the second quality check is performed by service A in one day but at a cost of 3,000 euros. The initial 1,000-euro 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.

Within the Discorso project, we’ve developed an advanced framework for the design and enactment of flexible and dynamic business processes. From a research standpoint, we’ve planned different enhancements. We plan to extend our models to incorporate exception handling, and to integrate it with monitoring capabilities. We’re also planning to extend service selection by reducing the optimization overhead to facilitate the management of multiple...
instances of the same business process. As for monitoring, we would like to investigate constraints on the mutual correctness of different processes, into a single and coherent framework.

Acknowledgments
The Italian FAR Project Discorso (Distributed Information Systems for Coordinated Service Oriented Interoperability) partially supported the work presented here.

References

ABOUT THE AUTHORS

DANIL0 ARDAGNA is an assistant professor of software engineering in the Department of Electronics and Information at the Politecnico di Milano. His research interests include Web services composition, automatic computing, and computer system costs minimization. Ardagna has a PhD in computer science from the Politecnico di Milano. Contact him at danilo.ardagna@polimi.it.

LUCIANO BARESI is an associate professor in the Department of Electronics and Information at the Politecnico di Milano. His research interests include dynamic software systems, service-oriented applications, and software architectures. Baresi has a PhD in computer science from the Politecnico di Milano. He’s an editorial board member of Service Oriented Computing and Applications. Contact him at luciano.baresi@polimi.it.

SARA COMAI is an associate professor in the Department of Electronics and Information at the Politecnico di Milano. Her research interests include graphical query languages and notations and the specification, design, and automatic generation of complex Web applications. Comai has a PhD in computer science from the Politecnico di Milano. She coauthored Designing Data-Intensive Web Applications (Morgan Kaufmann, 2002). Contact her at sara.comai@polimi.it.

MARCO COMUZZI is an assistant professor in the School of Industrial Engineering at the Eindhoven University of Technology. His research interests include quality of service in service-oriented architectures and contract monitoring and auditing in electronic business networks. Comuzzi has a PhD in computer science from the Politecnico di Milano. Contact him at m.comuzzi@tue.nl.

BARBARA PERNICI is a full professor of information systems in the Department of Electronics and Information at the Politecnico di Milano. Her research interests include workflow information systems design, cooperative information systems, adaptive information systems, service engineering and Web services, data quality, and computer-based design support tools. Pernici has an MS in computer science from Stanford University. She served as elected chair of both TC8 Information Systems and IFIP WG 8.1. Contact her at barbara.pernici@polimi.it.