PAWS: A Framework for Executing Adaptive Web-Service Processes

Danilo Ardagna, Marco Comuzzi, Enrico Mussi, Barbara Pernici, and Pierluigi Plebani, *Politecnico di Milano*

The Processes with Adaptive Web Services framework couples design-time and runtime mechanisms to flexibly and adaptively execute managed Web-services-based business processes.

Researchers in autonomic computing envision self-managed applications capable of supporting self-configuring, self-optimizing, self-healing, and self-protecting computing systems.[1] Recent applications of this vision to business processes based on Web services (hereafter, *services*) are improving flexibility and adaptivity in the service-oriented approach.

We define a *flexible* process as one that can change its behavior dynamically according to variable execution contexts; an *adaptive* process is one that can execute a service when conditions at runtime differ from those assumed during the service's initial design. While researchers have proposed several adaptation mechanisms, none of the existing frameworks systematically couples adaptation design-time and runtime execution. Focusing only on design-time issues reduces an application's ability to adapt its behavior at runtime. In a loosely coupled environment, in fact, the actual execution context might differ dramatically from the conditions hypothesized during application design. In addition, implementing advanced process-execution mechanisms, such as service substitution, is possible only when processes and services are carefully designed in advance.

To address this, we developed PAWS (Processes with Adaptive Web Services), a framework for flexible and adaptive execution of managed service-based processes. Our framework coherently supports both process design and execution. It also integrates several research results developed at Politecnico di Milano that address different aspects of adaptation, coupling design-time and runtime mechanisms in a global environment.

We have two primary goals for PAWS. First, we want it be self-optimizing. PAWS should select the best available services for executing the process and define the most appropriate quality-of-service (QoS) levels for delivering them. Second, PAWS should guarantee service provisioning, even in case of failures, through recovery actions and self-adaptation if the context changes. To meet these goals, PAWS provides methods and a toolset to support design-time specification of all information required for automatic runtime adaptation of

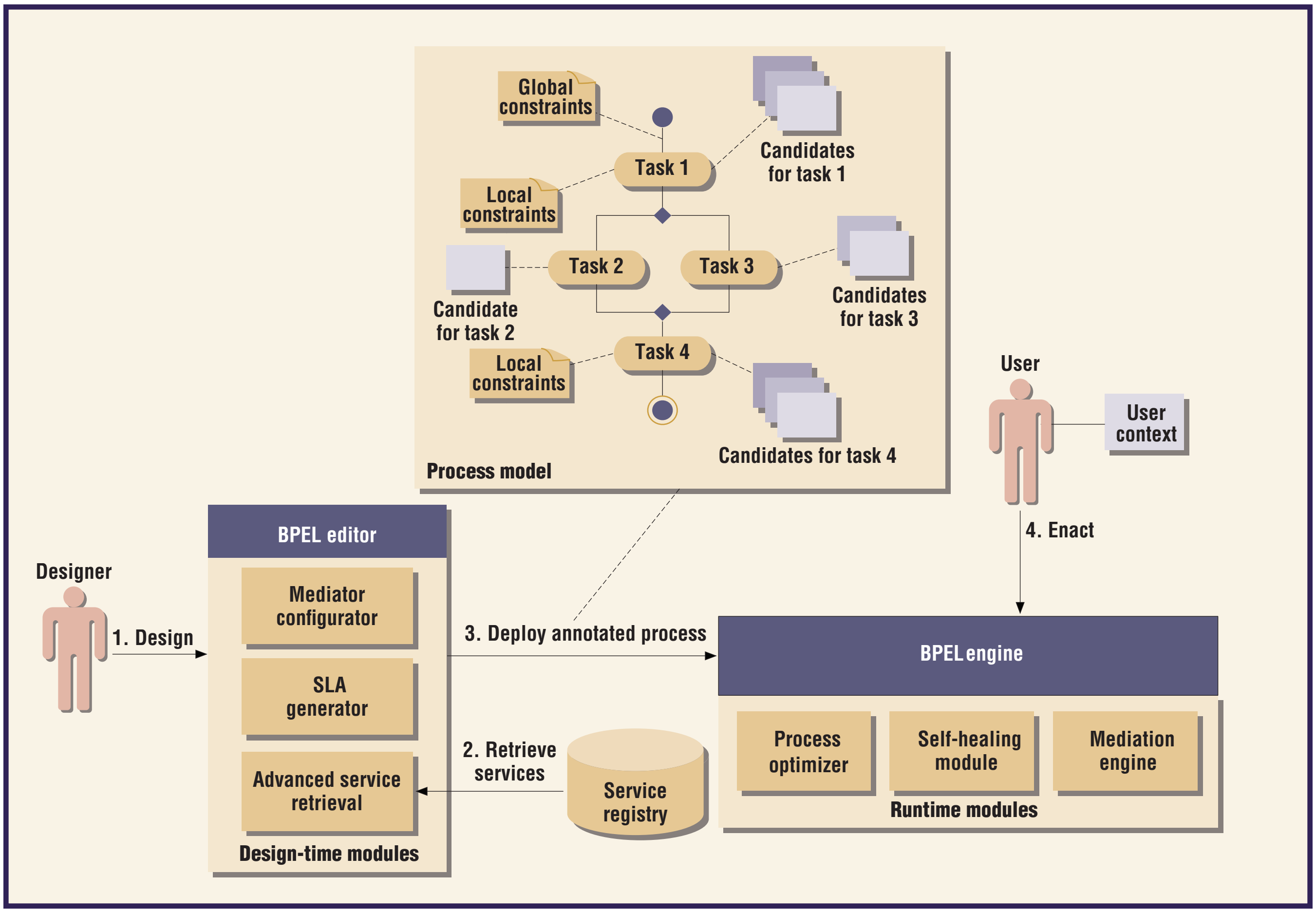

Figure 1. The Processes with Adaptive Web Services architecture. PAWS consists of design-time and runtime modules. Using a Business Process Execution Language editor, designers initially focus on process definition, then select services to perform the required invocations and satisfy the annotation constraints.

processes according to dynamically changing user preferences and context.[2-4] Here, we focus on how PAWS selects and adapts candidate services for a composed process. In addition, we describe its annotations for exploiting runtime flexibility. PAWS provides flexibility in terms of optimization, mediation, and self-healing functionalities. Our framework is general; developers can extend it by introducing domain-dependent service annotations with QoS and context definitions.

The PAWS framework

Figure 1 shows the design-time and runtime modules that constitute the PAWS framework.

In PAWS, both service discovery and service selection are driven not only by functional aspects (what the service should do), but also by nonfunctional aspects (how the service should work). Regarding the latter, all PAWS modules rely on a shared QoS model[5] to express global and local QoS constraints, to discover the set of candidate services, and to select the most suitable services.

The QoS model's goal is to provide an extensible way to express the quality of tasks and the overall process. A QoS dimension represents a specific quality aspect, such as response time, cost, or availability. The QoS model specifies a quality dimension using

- a name,
- a metric,
- a range of admissible values (either interval or categorical),
- a utility function that states how the QoS varies with respect to the dimension's assumed values, and
- any domain-dependent QoS properties.

We assume that a service's quality and functional description is published in the service registry.

Design-time modules

Design-time modules let designers create the annotated BPEL (Business Process Execution Language) process specification. Starting from a standard BPEL editor, designers can define both global and local constraints. A constraint defines a QoS or domain-dependent requirement for each task. By annotating the BPEL process, the designer couples the BPEL specification's functional requirements with the PAWS constraints, which usually refer to nonfunctional aspects. For example, a designer might require a given task to be performed in a given time or the overall cost to be within a given budget.

In traditional design approaches,[3] designers start by identifying potential services and then define the BPEL process using these previously selected services. In contrast, with PAWS, designers can initially focus on the process definition (step 1 in figure 1) and then select services to perform the required invocations and satisfy the annotation constraints. So, given a specific task, in step 2, the designer relies on the *advanced service-retrieval* module to retrieve all published services that can perform the task. The retrieval module does this by comparing the required service interface—defined in Web Services Description Language (WSDL)—with the published ones. It also verifies that the service's QoS satisfies the local constraints. Services that pass both analyses constitute the candidate set for the target task.

From a functional perspective, the retrieval process returns services similar to the desired one; rarely does a service's WSDL interface match exactly. We therefore need mediators to translate between the two service-interface signatures. To this end, PAWS includes a *mediator configurator* to support set-up of the related runtime module—the *mediator engine*. If it's not possible to derive message transformations automatically, the designer defines them during process execution, providing additional information about parameter and service mappings.

Regarding nonfunctional constraints, the service-retrieval module includes a service in the candidate set if it can support the required QoS. Before invoking the service, the designer uses the *SLA (service-level agreement) generator* to define the actual QoS that the service commits to support during process execution and that PAWS will monitor at runtime. Finally, in step 3, PAWS deploys the annotated BPEL process.

Runtime modules

The PAWS process definition doesn't include invocations of real services, but rather of desired services as characterized by the annotated process. Once the user starts executing the deployed process (step 4 in figure 1), the process optimizer runtime module selects a service for each task from among the candidates. Moreover, because a traditional BPEL engine executes the process and the target services are selected at runtime, runtime modules mediate between the BPEL invocation—as specified in the process definition—and the selected service's invocation.

The *process optimizer* module selects the service, aiming both to satisfy the annotations' local and global QoS constraints and to maximize overall quality for the user. The user context collects user preferences and transparently exchanges context information between the user and the framework. PAWS represents context by `<name,value>` pairs, and can thus define generic and domain-dependent characteristics, such as user-defined QoS priorities, negotiation preferences, and user location information. (We discuss PAWS context management in detail elsewhere.[6])

PAWS mediates services using the *mediation engine* module, which the *mediator configurator* module sets up at design-time. The mediation engine redirects invocation of the deployed process to the proper selected services.

As long as the process execution works, no additional efforts are required. However, if a participating service or the overall process fails, the *self-healing* module handles adaptation accordingly. When it detects a fault during process execution, the self-healing module implements a set of semiautomatically managed recovery actions that enhance process adaptivity. If the failure requires a service substitution for a given task, the process optimizer module selects services for the remaining tasks from the candidate set to guarantee global constraints.

PAWS lets designers initially focus on the process definition, then select services.

Components for flexibility and adaptivity

We now describe how PAWS module functionality achieves the key requirements for process adaptivity and flexibility. The sidebar, "Related Work in Web-Service-Based Business Processes," describes other efforts in this area.

Advanced service retrieval

PAWS performs service retrieval through

Related Work in Web-Service-Based Business Processes

Research on adaptation and flexibility in business processes includes many heterogeneous areas, from traditional research on the service-oriented architecture (SOA) to Semantic Web services and business process optimization.

Michael Papazoglou and colleagues advocate the need to extend the traditional SOA to consider service composition and service management.[1] Other researchers propose adaptive mechanisms for workflows and service-based processes. The Mosaic framework[2] proposes to model, analyze, and manage service models focusing on design phases rather than runtime flexibility. Meteor-S[3] and other semantic-based approaches—such as the Web Services Modeling Ontology (www.wsmo.org)—explicitly define the process goal of both service discovery and composition. The Meteor-S approach proposes a framework to select semantically annotated services that focus on flexible process composition and QoS properties. However, it doesn't consider runtime adaptivity to react to changes and failures. The WSMO approach proposes a goal-based framework to select, integrate, and execute Semantic Web services. It doesn't separate design and runtime phases, nor does it specifically support adaptivity. While goal-based approaches can make it possible to derive service compositions at runtime, their applicability in open service-based applications is limited by the amount of knowledge available in the service definitions.

Other approaches tackle individual aspects of adaptation.[4,5] Business-process-optimization approaches provide the process specification and select the set of best services at runtime by solving an optimization problem.[4] Similarly, researchers model grid systems applications as high-level scientific workflows that select resources at runtime to minimize, for example, workflow execution time or cost. Even if such approaches perform runtime re-optimization and provide a basic adaptation mechanism, they have a limited ability to address user context changes and self-healing.

PAWS supports adaptation preferences and adaptation execution in a coherent framework, focusing on both functional and QoS properties. With respect to self-healing, the PAWS framework allows runtime adaptation and flexibility based on optimization and negotiation mechanisms and predefined repair actions.

References

1. M.P. Papazoglou and D. Georgakopoulos, "Service Oriented Computing: Introduction," *Comm. ACM*, vol. 46, no. 10, 2003, pp. 24–28.
2. B. Benatallah et al., "Service Mosaic: A Model-Driven Framework for Web Services Life-Cycle Management," *IEEE Internet Computing*, vol. 10, no. 4, 2006, pp. 55–63.
3. J. Cardoso and A.P. Sheth, "Semantic E-Workflow Composition," *J. Intelligent Information Systems*, vol. 21, no. 3, 2003, pp. 191–225.
4. L. Zeng et al., "QoS-Aware Middleware for Web Services Composition," *IEEE Trans. Software Eng.*, vol. 30, no. 5, 2004. pp. 311–327.
5. J.O. Kephart and D.M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, 2003, pp. 41–50.

URBE (UDDI Registry By Example), a UDDI extension that supports content-based queries. Unlike current registries, which limit service location options to browsing predefined taxonomies or searching keywords,[7] URBE also lets designers submit a WSDL specifying the requested service in terms of supported operations and I/O parameters. URBE then returns a set of published services that more closely match the requested WSDL interface.[8]

URBE is composed of a functional and a QoS matchmaker. The functional matchmaker is driven by a similarity-evaluation algorithm that can state the similarity among services on the basis of their descriptions. The algorithm starts with the assumption that two services are equal if they both require the same input information and produce the same output data for the same operations. The similarity algorithm works with WSDL service signatures and accounts for

- naming aspects, which refer to the names adopted for identifying the service, available operations, and related exchanged parameters; and
- structural aspects, which refer to the number of operations available and the I/O parameters' data types.

The algorithm compares names using an ontology that organizes terms according to semantic relationships, such as synonym, antonym, homonym, and hypernym.[8] By adopting a general-purpose ontology, such as WordNet, the algorithm lets designers add domain-specific terms and relationships. The algorithm can also process Semantic Annotations for WSDL and XML Schema (www.w3.org/TR/sawsdl) descriptions, which allows for specific semantic annotations on WSDL elements.

If the designer requirements include QoS constraints, PAWS invokes the QoS matchmaker after the functional one. Using the PAWS QoS model, the matchmaker verifies whether a request and an offering match—

that is, whether the offering can support the request. At this stage, we use WS-Policy as the language for QoS offerings.

Researchers validated URBE's similarity algorithm in experimental evaluations; its precision and recall in specific application domains were around 80 percent.[9]

Negotiation

The SLA generator allows PAWS to automatically negotiate QoS aspects between the user and candidate service providers. The SLA generator negotiates the QoS of candidate services only for tasks with local budget constraints. Indeed, such constraints can be solved at design-time—that is, the user and provider can negotiate the maximum QoS possible within the user's budget constraint. Automating negotiation at design-time saves time and simplifies the runtime process optimization phase. Therefore, for each task that specifies a local budget constraint, the SLA generator negotiates the SLA for all the candidate services URBE retrieves.

The negotiation's objective is to obtain an SLA within the request and service-offering matches identified in the service-discovery phase. Specifically, given a QoS dimension range, our QoS model lets designers define a set of negotiation intervals over which service providers can specify a pricing model. The pricing model is additive—the final price associated to a service is the sum of partial prices associated to each individual QoS dimension interval. The SLA generator uses the designer's local-constraints budget during negotiation to improve candidate service quality, starting from a basic SLA identified by each relevant QoS dimension's lowest QoS interval. The module can adopt different strategies, such as splitting the budget proportionally to the users' priorities (a horizontal strategy) or exploiting the budget to maximize the highest priority dimension's QoS (a vertical strategy).

Negotiating SLA for a single candidate service exploits two configuration policies: that of the service provider and that of the designer (who acts on the user's behalf). The user policy contains the QoS dimension preferences (expressed as a vector of weights) and identifies the SLA negotiation strategy. The service provider policy contains the target service's pricing model parameterization. After parsing both policies, the SLA generator can automatically perform SLA negotiation.

If a feasible solution to the process optimization problem doesn't exist, PAWS can update the negotiated QoS profiles at runtime using the process optimizer module. Runtime negotiation exploits

- an extra budget obtained from the global budget constraint, and
- a structure of preferences on QoS dimensions and service pricing models similar to those specified for design-time SLA negotiation.

To evaluate these two negotiation strategies, we ran an experiment, defining quasilinear utility functions for users and providers with different combinations of budget constraints and pricing models. The horizontal negotiation strategy's outcome proved to be extremely close to the negotiation problem's Pareto frontier. This result is noticeable because the negotiation strategies are heuristics and we assume incomplete information among negotiation participants.[10,11]

Business process optimization

Given each task's set of candidate services, we model service selection as an optimization problem that accounts for end-user preferences, global QoS process constraints, and the runtime execution context. Furthermore, we interleave service selection and execution: optimization occurs when the user instantiates and executes the business process, and it iterates during process execution. This reoptimization accounts for service performance variability, invocation failures, and user context changes.

Researchers have proposed several business process optimization approaches. Some guarantee global constraints for only the critical path—that is, the path that corresponds to the highest execution time[12]—while others satisfy global constraints only statistically.[13] Furthermore, in such approaches, if the end user introduced severe QoS constraints for the composed service execution, such as a stringent execution time limit, the system wouldn't find a solution and the service execution would fail.

In PAWS, we implemented a new optimization approach based on mixed-integer linear programming models. This approach overcomes the limits of previous solutions when optimizing user-perceived QoS under severe

The SLA generator supports automatic QoS negotiation between users and service providers.

We conceive of self-healing behavior as a combination of monitoring and repair capabilities.

QoS constraints. Our modeling approach is based on

- loops peeling, which considers the probability distribution of the number of loop iterations; and
- negotiation, to bargain QoS parameters with service providers when feasible solutions can't be found, thus reducing process invocation failures.

The optimizer ranks each task's set of candidate services (excluding those that violate constraints). It thereby selects a service for each task and ranks other services as substitutes that can be invoked in case of a failure.

Experiments have shown that our joint optimization and negotiation approach is effective, particularly for large processes (up to 10,000 tasks) with severe QoS constraints. Our approach also reduces the re-optimization overhead.[14] In relation to existing approaches, our problem formulation and algorithms improved the end user's QoS by up to 40 percent.

Mediation support

Our mediation engine aims to

- support service invocation, dynamically binding a generic candidate service without requiring stub compilation at design time, and
- manage service substitution, which might involve services that are described by different signatures but have the same choreography.[15]

Whenever the BPEL engine invokes a task, the mediator selects the first service from the optimizer's ranked candidate-services list. If the candidate service's interface differs from the interface that the task definition requires, the mediator retrieves the proper mapping document produced by the mediator configurator, and then invokes the candidate service by sending transformed messages. The mediator manages candidate-service invocation through sessions. By so doing, it avoids the need to repeatedly access the ranked list if a service is executed more than once in several tasks. The sessions also allow stateful service execution. During service substitution, the mediator uses the mediator configurator to transform messages formatted according to the substituted service's interface to suit that of the substitute service.

Self-healing

We conceive of self-healing behavior as a combination of monitoring and repair capabilities. Our goal is to detect failure during a process execution and apply appropriate recovery actions to let the process successfully terminate.

To that end, PAWS features several recovery actions:

- *retry* a process task's execution,
- *redo* a process task's execution using different input parameters,
- *substitute* the current candidate service with another candidate service, and
- *compensate* an executed task with a compensation action defined within the service management interface.

Our self-healing module uses the mediator to perform recovery actions, either to retry/redo process task execution or to substitute a faulty candidate service. In case of service substitution, the mediator first checks for changes in the user's context since the last optimization. If changes have occurred, it notifies the optimizer, which then generates a new ranked candidate-services list. If no changes have occurred, the mediator retrieves the next candidate service from the optimizer's list and waits for the self-healing module to restart the process. In both cases, if a candidate service is no longer available, the mediator throws an exception.

The self-healing module typically uses the retry and the redo actions to recover from temporary faults, while it uses the substitute action when an orchestrated service instance is considered permanently faulty. It uses compensation actions to restore a previous process state. Because the self-healing module can't perform recovery actions over running-process instances, once a fault is detected, it suspends the process and moves it into a repair mode. Once all necessary recovery actions are complete, the instance resumes and the self-healing module returns the process to the running mode.

To perform the retry and the redo actions, the self-healing module explicitly asks the mediation engine to execute an already-executed process task; it performs substitution by asking the mediation engine to substitute a faulty candidate service with a new one.

The WS-Diamond Project (http://wsdiamond.

di.unito.it) has tested our self-healing and mediation modules, executing a repair plan using a diagnoser and a plan generator.[16]

Scenarios and applicability criteria

We implemented the PAWS framework in Java, using

- Axis handlers to realize the mediation engine,
- an extended Java implementation of UDDI (jUDDI) version to provide service retrieval functionalities,
- WordNet as a semantic network for term-based similarity evaluations,
- ActiveBPEL as a BPEL engine (properly extended with plug-ins for managing repair actions),
- Web Services Distributed Management to provide notification services to support process management operations, and
- CPLEX as an optimization engine.

We implemented all PAWS modules as services themselves. As a result, developers can use them in different combinations to support flexibility and adaptivity. (PAWS modules are available as open source code upon request.)

Researchers have tested the PAWS framework in multiple contexts to build adaptive information systems based on Web services. The Multichannel Adaptive Information Systems (MAIS) project (www.mais-project.it) features two proof-of-concept prototypes:

- a virtual travel agency, which sells travel packages to customers through multiple system channels, and
- a micro MAIS application that gathers information at archeological sites following natural disasters using networked PDAs.

WS-Diamond focuses on the development of service-based self-healing systems. The reference scenario is a food company that cooperates with several partners to sell food packages. The E-Adaptive Services for Logistics project (www.easylog.org) aims to support risk management in dangerous-goods transportation. Finally, the Distributed Information Systems for Coordinated Service Oriented Interoperability project (www.discorso.eng.it) seeks to develop flexible processes to support small and medium-sized enterprises.

In all these projects, the developed applications require a basic adaptivity layer. This layer consists of URBE services and the mediation engine, which selects services and adapts the process execution. The mediation configurator has been adopted in both the MAIS and WS-Diamond projects, in which the developed candidate services have different functional interfaces. Projects with critical, global QoS constraints have adopted the process optimizer, while projects that can accept a range of possible QoS values, such as MAIS, Discorso, and Easylog, use the SLA generator. Finally, WS-Diamond introduced the self-healing module to provide fault-tolerant environments.

These projects have shown that the PAWS framework reduces design-time efforts to create a flexible process, while the quasi-Pareto optimality ensures a good trade-off between the user and provider perspectives. In addition, our framework is based on open standards and can easily fit in existent architectures. At this stage, PAWS' main limitation is performance; the modules introduce a noticeable overhead at runtime. The results also show that PAWS has limited applicability in mobile environments because of its computing and power requirements.

We've planned several enhancements to the PAWS framework. Regarding self-configuration, we aim to extend the mediation capabilities to reduce the design-time effort and automate mediation configuration as much as possible, especially for complex data structures. Moreover, we want to strengthen adaptivity using new and more flexible context-management mechanisms to express user and provider preferences on the QoS of the composed process. In addition to our mediation-layer work, we plan to enhance adaptivity by extending negotiation functionalities through more detailed user-provider contract descriptions.

We're also planning to extend PAWS' self-optimizing aspect in two ways. First, we want to extend self-optimization to account for fault probability. Second, we intend to reduce optimization overhead and thus facilitate optimization of multiple process instances. To enhance self-healing, we will introduce planning capabilities to recover orchestrated processes using atomic recovery actions both individually and

in combination.[16] Service descriptions also affect autonomic systems' self-configuring, self-optimizing, self-healing, and self-protecting properties; we therefore plan to extend Web service specifications to enhance our service-retrieval approach.

See http://home.dei.polimi.it/pernici/ws-research.html for further information on PAWS.

Acknowledgments

This work was supported by the European Commission's Information Society Technologies Future and Emerging Technologies—Specific Targeted Research Project, WS-Diamond; the Italian Ministry of Research and University's MAIS Funds for Basic Research project; the Discorso and Easylog Funds for Applied Research projects; and the Research Programs of Relevant National Interests' Quadrantis (Quality of Service, Diagnosis, Reaction, Assessment Techniques in Information Systems) project.

References

1. J.O. Kephart and D.M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, 2003, pp. 41–50.
2. Z. Maamar, S.K. Mostéfaoui, and H. Yahyaoui, "Toward an Agent-Based and Context-Oriented Approach for Web Services Composition," *IEEE Trans. Knowledge and Data Eng.*, vol. 17, no. 5, 2005, pp. 686–697.
3. M.P. Papazoglou and W.-J. van den Heuvel, "Web Services Management: A Survey," *IEEE Internet Computing*, vol. 9, no. 6, 2005, pp. 58–64.
4. B. Pernici, "Service Design and Development," *Service Oriented Computing*, Dagstuhl Seminar Proc., Internationales Begegnungs und Forschungszentrum fuer Informatik (IBFI), 2005; http://drops.dagstuhl.de/opus/volltexte/2006/525.
5. C. Cappiello, M. Comuzzi, and P. Plebani, "On Automated Generation of Web Service Level Agreements," *Proc. Int'l Conf. Advanced Information Systems Eng.* (CAISE 07), LNCS 4495, Springer, 2007, pp. 264–278.
6. C. Cappiello et al., "Context Management for Adaptive Information Systems," *Int'l Workshop Context for Web Services* (CWS 05), Elsevier, 2005, pp. 264–278.
7. J. Garofalakis et al., "Contemporary Web Service Discovery Mechanisms," *J. Web Eng.*, vol. 5, no. 3, 2006, pp. 265–290.
8. D. Bianchini et al., "Ontology-Based Methodology for e-Service Discovery," *J. Information Systems*, vol. 31, nos. 4–5, 2006, pp. 361–380.
9. P. Plebani and B. Pernici, *Web Service Retrieval Based on Signatures and Annotations*, tech. report 2007.47, Dept. Electronics and Information, Politecnico di Milano, July 2007.
10. M. Comuzzi and B. Pernici, "An Architecture for Flexible Web Service QoS Negotiation," *Proc. Int'l Enterprise Distributed Object Computing Conf.* (EDOC), IEEE CS Press, 2005 pp. 70–82.
11. N. Jennings et al., "Automated Negotiation: Prospects, Methods and Challenges," *Group Decision and Negotiation*, vol. 10, no. 2, 2001, pp. 199–215.
12. L. Zeng et al., "QoS-Aware Middleware for Web Services Composition," *IEEE Trans. Software Eng.*, vol. 30, no. 5, 2004, pp. 311–327.
13. J. Cardoso and A.P. Sheth, "Semantic E-Workflow Composition," *J. Intelligent Information Systems*, vol. 21, no. 3, 2003, pp. 191–225.
14. D. Ardagna and B. Pernici, "Adaptive Service Composition in Flexible Processes," *IEEE Trans. Software Eng.*, vol. 33, no. 6, 2006, pp. 369–384.
15. V. De Antonellis et al., "A Layered Architecture for Flexible Web Service Invocation," *Software—Practice and Experience*, vol. 36, no. 2, 2006, pp. 191–223.
16. L. Console et al., *WS-Diamond: An Approach to Web Services, Diagnosability, Monitoring, and Diagnosis*, tech. report 2007.57, Dept. Electronics and Information, Politecnico di Milano, 2007.

About the Authors

Danilo Ardagna is an assistant professor of information systems at the Department of Electronics and Information, Politecnico di Milano. His research interests include Web service composition, autonomic computing, and computer system costs minimization. He received his PhD in computer engineering from Politecnico di Milano. Contact him at Politecnico di Milano, Piazza Leornardo Da Vinci 32, 20133 Milan, Italy; ardagna@elet.polimi.it.

Marco Comuzzi is a research associate at Politecnico di Milano. His research interests focus on automated negotiation algorithms in the context of Web service quality. He has a PhD in computer engineering from Politecnico di Milano. Contact him at Politecnico di Milano, Piazza Leornardo Da Vinci 32, 20133 Milan, Italy; comuzzi@elet.polimi.it.

Enrico Mussi is a research associate at Politecnico di Milano. His research interests include context-based and self-healing compositions of Web services and service mediation. He received his PhD in computer engineering from Politecnico di Milano. Contact him at Politecnico di Milano, Piazza Leornardo Da Vinci 32, 20133 Milan, Italy; mussi@elet.polimi.it.

Barbara Pernici is a professor of computer engineering at Politecnico di Milano. Her research interests include cooperative information systems, service management, workflow management systems, and information systems modeling and design. She has an MS in computer science from Stanford University and a Laurea Degree in engineering from Politecnico di Milano. She is an editor of the *ACM Journal of Data and Information Quality*, the *Requirements Engineering Journal*, and the *International Journal of Cooperative Information Systems*. She is chair of the International Federation for Information Processing's Technical Committee 8 (TC8, Information Systems) and Working Group 8.1 (Information Systems Design). Contact her at Politecnico di Milano, Piazza Leornardo Da Vinci 32, 20133 Milan, Italy; barbara.pernici@polimi.it.

Pierluigi Plebani is a research associate at the Dipartimento di Elettronica ed Informazione of Politecnico di Milano. His research interests include Web services technologies, such as service-retrieval methods, as applied to multichannel information systems; data quality in cooperative information systems; and information systems security assessment. He has a PhD and a Laurea Degree in computer science from Politecnico di Milano. Contact him at Politecnico di Milano, Piazza Leonardo Da Vinci 32, 20133 Milan, Italy; plebani@elet.polimi.it.