Using Policies to Manage Composite Web Services

When no single Web service can fulfill a user’s request, providers often turn to composite Web services—combinations of Web services. Developing a composite Web service is not trivial, however. It requires a specification that shows elements such as the component Web services’ execution order, the data dependencies among the component Web services, and corrective strategies if the component Web services raise exceptions. Deploying the composition specification at runtime consists of identifying and triggering the component Web services, overseeing their execution, coordinating their actions to avoid conflicts, managing their dependencies, and triggering corrective strategies for exception handling. Specifications for Web services composition include the Web Services Flow Language (WSFL), the Business Process Execution Language (BPEL), and the Web Services Composite Application Framework (WS-CAF).

Most researchers agree that a Web service’s engagement in a composite Web service is a given, to a certain extent—that is, engagement doesn’t depend on the Web service’s execution state. However, it need not always be so (“What Can Context Do for Web Services?” Z. Maamar, D. Benslimane, and N.C. Narendra, to be published, Comm. ACM, 2006).

Indeed, before agreeing to engagement, a Web service must

- consider its internal execution state,
- question whether it would be rewarded for the engagement,
- know how to manage its performance if it gets disrupted,
- know how to secure its peer interactions,
- know how to adapt its behavior in case of failure, and
- know when or when not to participate in a composite Web service and when to recommend peers that might better satisfy an engagement request.

Accounting for all these aspects without establishing a Web service’s behavior is challenging at best. Guidelines are needed to back a Web service’s correct execution. These must be stated and checked when the Web service is designed. Additional guidelines are needed to overcome semantic disparities that may arise between heterogeneous Web services. Finally, guidelines must specify what constitutes an acceptance failure of a Web service, which can vary from one business case to another. The various guidelines can be mapped onto policies, which would in essence define a Web service’s acceptable behavior (“Developing Interoperable Business Processes Using Web Services and Policies,” Z. Maamar and colleagues, to be published Proc. 2nd Int’l Conf. Interoperability for Enterprise Software and Applications, Springer, 2006).

WHY POLICIES?

One definition of policies views them as information that developers can use to modify a system’s behavior (“Conflicts in Policy-Based Distributed...
WEB SERVICES

Figure 1. Six scenarios that illustrate potential uses of policies in Web services: announcement (a), selection (b), compatibility (c), agreement (d), verification (e), and composition (f).

Systems Management,” E.C. Lupu and M. Sloman, IEEE Trans. Software Eng., vol. 25, no. 6, Nov.-Dec. 1999, pp. 852-869). Our definition goes beyond behavior modification and considers policies as external, dynamically modifiable rules and parameters that are input into a system. The system can then adjust to administrative decisions and changes in the execution environment. Thus, in our definition, policies specify actions that must be performed according to detected events and context changes.

Policies so defined are particularly appropriate for Web services composition. Developers and deployers can use them to manage Web services at both high and low levels. High-level policies would address composition, including what user preferences to integrate into Web services, what is required to satisfy these preferences, and when to track Web services’ execution progress. Low-level policies would include how to exchange comprehensible information among Web services, how to deal with Web service unreliability, how to substitute a Web service with another peer without disrupting the execution flow, and how to suspend a Web service execution because of risk that behavior might change or that information could be intercepted.

Adopting policies also introduces the possibility of changing Web services’ behavior without altering a composition specification. By using policies, Web service deployers can continuously adjust multiple aspects such as Web services’ conflict-resolution mechanisms to accommodate variations in the environment. Finally, policies would be useful in tracking actions that cause unexpected exceptions and in specifying how these exceptions are to be repaired without the need to alter the composition specification itself.

ASSOCIATING POLICIES WITH SERVICES

There are many ways to associate policies with Web services. Figure 1 illustrates six possible scenarios.

The announcement scenario (Figure 1a) shows how offered and required policies, in terms of rules, can be part of the Web services descriptions in the Web Services Description Language (WSDL) posted on an UDDI-like registry. These policies dictate how a Web service acts as a component in a composite Web service.

Next, the selection scenario (Figure 1b) shows how to use policies to identify a Web service or a community of Web services, to satisfy a consumer request. The consumer could be a user or another Web service. Here, the consumer is interested in only the Web services that offer policy P_c.

The compatibility scenario (Figure 1c) shows how interactions among Web services can also happen at the policy layer.
level, which calls for guaranteeing the conformance of policies with each other. This conformance extends to the data that policies manage and exchange.

Next, the agreement scenario (Figure 1d) shows how a consumer, such as a Web service, and another Web service agree on the policies to use, namely $P_C$ and $P_E$. Negotiation and compatibility actions might precede the agreement.

The verification scenario (Figure 1e) extends the agreement scenario. An agreement is mainly a promise to conform to some policies, for instance $P_C$ and $P_E$. This promise must be fulfilled at runtime; otherwise, the Web service could be subject to penalties like lowering its reputation level. In Figure 1e, the interaction between Web services is a business transaction message subject to policies $P_C$ and $P_G$. The receiving Web service must verify that the message’s contents and other behaviors associated with this transaction conform to the agreed-on policies $P_C$ and $P_E$.

Finally, the composition scenario (Figure 1f) shows the composition of multiple Web services to meet the requirements of a single consumer. For example, suppose a consumer requires three policies, $P_A$, $P_B$, and $P_C$, and no single Web service offers them all. Two Web services, one that offers $P_A$ and $P_B$ and another that offers $P_C$, form a composite Web service that jointly offers $P_A$, $P_B$, and $P_C$.

**DEVELOPING STANDARDS FOR POLICIES**

The industry currently lacks a standard framework for enhancing Web services through policies. Two kinds of approaches—Semantic Web and Boolean combinations of policy predicates—could help compensate for this gap.

The Semantic Web approaches, such as KAoS (“KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction, and Enforcement,” A. Uszok and colleagues, *Proc. IEEE 4th Workshop Policies for Distributed Systems and Networks*, IEEE CS Press, 2003, pp. 93-98) and Rei (“A Policy Language for Pervasive Computing Environment,” L. Kagal, T. Finin, and A. Joshi, *Proc. IEEE 4th Workshop Policies for Distributed Systems and Networks*, IEEE CS Press, June 2003, pp. 63-76), have not gathered significant industry support to date, although they might be important in handling certain types of policies in the future. A Semantic Web-driven policy could relate high-level management and regulatory policies to low-level implementation and enforcement-level policies. For example, suppose a high-level policy requires that personal information from customers be deleted if not used for 90 days. Taxonomies and ontologies can enrich this policy by stating that customer name, email address, and postal address...
WEB SERVICES

are personal information, and these must be deleted if the difference between the date of last use and current date is greater than 90 days.

A Semantic Web-driven policy is also useful when two Web services must form a composite Web service once their semantic heterogeneities are resolved. At present, in contrast, Web service policies are used with agreed-on interoperability standards, where both parties necessarily bind to the same terminology, rules, and options. In fact, both parties operate under a contract to use a particular set of standards. Policies driven by a Semantic Web approach would merely determine which rules and options each Web service will use for what transaction.

Several proposals fall into the Boolean combination of policy predicates category, which does appear to have industry support. First, many proposed domain-specific policy standards are adopting the WS-Policy framework, the proposal with the most support among vendors. A second proposal, the Web services Policy Language (WSPL), developed by the Oasis eXtensible Access Control Markup Language (XACML) Technical Committee, provides services with the functions of policy framework, domain-specific policy assertions, and binding policies. However, WSPL currently isn’t progressing toward standard status in the XACML Technical Committee because it falls outside the scope of the committee’s charter, which is limited to “authorization and access control.” A third Boolean combinations proposal is to add Boolean operators to the WSDL 2.0 “Features and Properties” element, but the WSDL working group closed this proposal without action. A fourth proposal suggests a domain-independent language for use in expressing domain-dependent policy assertions in any Boolean policy framework (http://research.sun.com/projects/xacml/ws-policy-constraints-current.pdf). The Oasis standards body is discussing this proposal for inclusion in a possible standard.

APPLICATIONS FOR POLICIES

Policies can play specific roles in Web service exception handling and semantic heterogeneities. An exception is any event that could disrupt the normal execution progress of any application. BPEL, the well-known standard for defining and executing composite Web services, deals only with predefined exceptions, such as Web-service unavailability. Composite Web services are by nature distributed and are thus exposed to a wide variety of failures. It’s unrealistic to anticipate dealing with all possible failure types. Unexpected exceptions can occur during Web service execution—for example, high execution cost when low execution cost is expected, execution results don’t meet user demands, or noncommitment to some given policies (agreement and verification scenarios).

Exception handling

Because unexpected exceptions currently fall outside BPEL’s scope, an effective approach is to enrich a composite Web service’s specification with policies that let the composition run while handling the unexpected exceptions. This would be made possible by separating the policies that handle exceptions from the composite Web service’s code. Two policies will be required: declarative and event-condition-action (ECA). A declarative policy, such as WS-Policy, permits prior agreement on the recovery mechanisms to use per execution type. An ECA policy, such as Ponder, focuses on the type of exception that the Web service raises and the state of this Web service before carrying out actions. This leads to having rules of the form: Exception (X), WebServiceState(Y)_Action (Z), which means if exception X occurs when the Web service is in state Y, the Web service performs Action Z. Executing this rule will make the Web service transition to a new state.

Mediation

Because of the Internet’s heterogeneity, it’s unlikely that one provider would deliver all types of Web services. Tackling Web services’ semantic heterogeneity calls for mediation mechanisms. Mediation means the ability of two heterogeneous Web services to exchange meaningful and useful information regardless of their origins, execution chronology, communication protocols, and mapping or conversion techniques. The standard Web services protocol stack—Simple Object Access Protocol (SOAP), WSDL, and Universal Description, Discovery, and Integration (UDDI)—wasn’t initially intended to meet the requirements of a successful information exchange. The importance of achieving this exchange has triggered the development of many approaches (such as those discussed in “Semantic Interpretation and Matching of Web Services,” Chang Xu, Shing-Chi Cheung, and Xiangye Xiao Proc. 23rd Int’l Conf. Conceptual Modeling (ER’04), Springer, Nov. 2004, pp. 542-554; and “Inferring Data Transformation Rules to Integrate Semantic Web Services,” Bruce Spencer and Sandy Liu, Proc. 3rd Int’l Semantic Web Conf. (ISWC’04), Springer, Nov., 2004, pp. 456-470). Nevertheless, many challenges remain, especially those related to Semantic Web services.

Policies are a promising way to solve mediation issues in Web service composition scenarios. They could state how a composite Web service binds to a global ontology, how to establish mappings between component Web services’
local ontologies and the composite Web service’s global ontology, how to track various mappings for a successful mediation, and so on. Mediation policies would let the developer of a composite Web service engineer the data that component Web services exchange. This engineering depends on a specific ontology, involves type mapping and conversion, and targets input and output arguments of Web services. A mediation policy would for instance clearly state the currency that is associated with “price” argument and how it needs to be converted into other currencies.

Developers and deployers of composite Web services can use policies in various scenarios, from the announcement of Web services to the compatibility between Web services and composite Web services. Although policies offer much promise in enhancing Web services, there are caveats. If Web services become too flexible, with too many policy options, for example, Web service providers and consumers will be challenged in finding the Web services that are compatible with every requirement set. Our future work looks into the way policies would permit, first, managing business regulatory requirements and, second, defining the transactional properties as part of the operation of Web services.

**Zakaria Maamar** is an associate professor at Zayed University. Contact him at zakaria.maamar@zu.ac.ae.

**Djamal Benslimane** is a full professor at Université Claude Bernard Lyon 1. Contact him at djamal.benslimane@liris.cnrs.fr.

**Anne Anderson** is a senior staff engineer at Sun Microsystems Laboratories. Contact her at anne.anderson@sun.com.

For further information on this or any other computing topic, visit our Digital Library at http://www.computer.org/publications/dlib.