Toward Web Services Interaction Styles

E. Michael Maximilien
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120
maxim@us.ibm.com

Munindar P. Singh
North Carolina State University
Department of Computer Science
Raleigh, NC 27695
singh@ncsu.edu

Abstract

Service-Oriented Architectures (SOAs) are fundamentally changing the way in which we conceptualize and design business applications. An SOA-based application typically composes various distributed functions, including some possibly provided by external parties such as independent businesses. The key advantage of SOAs is the resulting dynamism, since the composed parts can be readily swapped out in favor of others of like functionality. SOA environments thus reflect the dynamism of human socioeconomic environments where businesses interact, collaborate, and expose services to each other in order to jointly create value. This paper presents a multiagent model for Web services and catalogs architectural styles that are key for SOA applications. It conceptually evaluates the styles by showing the kinds of service usages and the resulting dynamic interactions that they enable.

1. Introduction

Like real-world services, Web services embody relationships between entities who interact. Each Web service interaction involves at least a service provider and a service consumer. These relationships are reflected in the interactions between the participants. The interactions represent usage patterns and depend on the kinds of services offered. For instance, the usage of a service providing stock quotes requires in short-term interactions since the service consumer has only limited needs from the service—an input symbol results in a ephemeral stock quote value. At the opposite end of the longevity spectrum are interactions, such as purchase processes, which potentially last days (e.g., until goods are ordered and payments clear).

The above two interactions vary in terms of their duration. However, interaction styles also vary based on other criteria, such as how dynamic the intended interactions are, how cooperative the consumers are, how cooperative the providers are, and so on. The basic idea is that the types of interactions mimic business environments where autonomous parties provide services to each other and may form collaborative groups. For example, consumers in real-life settings can share quality ratings of the services and service providers that interest them to develop a sense of their evolving reputation, and thus to decide how much trust to place in them.

Since Web services represent functionalities exposed by different providers, they can be naturally characterized as autonomous, policy-driven agents [13, 16]. Viewing services as agents enables us to augment the interaction styles of Web services as interactions between and among service provider agents and service consumer agents. These styles are worth studying because they yield high-level characterizations of interactions, which would help us realize and implement superior SOA applications more effectively.

1.1. Contribution

Our primary contribution is in the creation of a catalog of interaction styles between and among consumers and providers of Web services. By augmenting the representation of service providers and consumers with software agents, we are able to capture a richer set of dynamic and autonomic interaction styles.

Our catalog forms the initial layout of a pattern language to discuss Web services interactions. This is in the spirit of software pattern languages [11, 15]; however, our patterns are different. Instead of reusable software components implementing a well-known and needed function, our patterns represent reusable interactions between consumers and providers of business functions to achieve a higher-level business solution to a business problem. For instance, by sharing monitored quality data, service consumer agents can help each other assess the trustworthiness of service providers and service implementations.

1.2. Organization

The remainder of this paper is organized as follows. Section 2 gives an overview of the background for our Web services agent-based framework. Section 3 gives an extensive catalog of different interactions styles in Web services, discussing each along with an abstract diagram. Section 4 gives a conceptual evaluation of the abstract interaction styles by giving concrete examples of current (or, sometimes, possible) usage in real-world applications. Section 5 discusses the related work in the worlds of Web services, multiagent systems, and software patterns. Finally, Section 6 presents some directions for future research.

2. Background

The vision of SOA calls for business functions to be modeled as Web services so as to enable a multitude of competitive service providers and implementations for each function, leading to a marketplace of services. In particular, a service consumer may interact with many providers for each service. A consumer may also interact with other consumers of that service (past and present). In essence, service consumers and providers form a virtual ecosystem.

The scale of the service market presupposes automation in dealing with it. For instance, the process of service discovery can be automated using semantic techniques [17]. To support that goal, overlaying a multiagent architecture on top of Web services gives us a platform in which to understand as well as automate Web services interactions.

We introduce a software service agent to each Web service used by a service consumer. The service agent exposes the interface exposed by the underlying service. Moreover, it exposes an additional agent-specific interface. Since service agents appear as Web services, they are easily incorporated into an application and minimize agent-specific changes in the application. We implemented this architecture as the Web Services Agent Framework (WSAF) [13]. Likewise, a similar agent can be attached to service providers and automate aspects of their interactions with consumers and with other providers. Service provider agents could learn about the consumers' preferences and provision their service implementations accordingly.

In addition to service consumer and provider agents, our multiagent architecture enables the creation of *rendezvous* nodes or *agencies* that enable agents to share information. Examples of agencies are those used for the collection of quality of service (QoS) data obtained from consumer agents and shared after service usage.

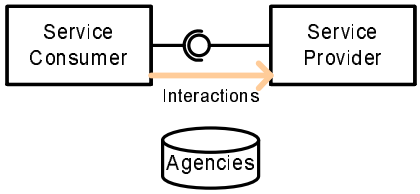

Figure 1. IS1: Direct consumer-to-provider interaction style. Consumers typically use service registries (at design-time) to discover and to select a service provider and a service implementation.

3. Interaction Styles

Using Garlan and Shaw's definition of architecture styles [12] as inspiration, we define Web services interaction styles as follows.

Definition 1 (Web Services Interaction Styles) *A family of interaction patterns reflecting interactions between and among consumer agents, provider agents, service implementations, and associated agencies intended to accomplish a high-level business goal of the service consumer and service provider. Each style represents specific constraints on the interactions of the parties involved.*

We begin with a catalog of interactions styles possible in an SOA and briefly show how our agent-based framework can be used to implement these styles. The styles incrementally build on each other by considering additional constraints and entities in the ecosystem. Section 4 evaluates the styles by describing classes of problems where they apply naturally.

We distinguish two primary classifications of consumer-to-provider interaction styles: simple interactions and agent-mediated interactions. Simple interactions occur directly between a consumer and a provider. Agent-mediated interactions use an intermediary agent [18, 7] that facilitates some aspects of the interaction. We concern ourselves primarily with agent-mediated interaction styles since they enable richer solutions. Table 1 summarizes the interaction styles.

With direct interactions, as depicted in Figure 1, the service consumers and service providers interact directly. Typically, each provider publishes its service interfaces and implementations in well-known registries. The consumer, at design-time, discovers a service interface, selects an appropriate implementation, and creates the necessary local proxies to bind to the service. In general, this interaction style is passive, since it occurs at design-time and a designer manually carries out the steps of interface discovery and implementation selection and binding.

No.	Style name	Agents' capabilities	Description
1	Direct	NA	Consumers directly interact with providers. At design-time, consumers use common registries to discover, select, and bind service providers and implementations
2	Consumer agent-mediated	Binding, selection, QoS monitoring, improve services, and negotiation	Consumers interact with providers using an intermediary agent. The consumer service agent automates some of the consumer's design-time and runtime tasks. Examples include agents used to bid on products on behalf of consumers, e.g., bidding agents on eBay
3	Provider agent-mediated	Add service functionality, provider collaboration and negotiation	Providers use intermediary agents to dynamically improve their service implementations and to dynamically collaborate with each other and present functionally richer services to consumers than otherwise possible
4	Consumer agent-mediated with referrals	Binding, selection, QoS monitoring, improve services, and consumer collaboration	Consumers interact with providers using an intermediary agent who in turn can also interact with other peer agents. Using referrals, consumer service agents collaborate to improve their tasks. An example is consumers collaborating to filter out products and services in online marketplaces, such as Amazon.com zShops' feedback system
5	Consumer and provider agent-mediated	Binding, selection, QoS monitoring, and consumer-provider collaborations and negotiations	Both consumers and providers use intermediary agents. The consumer and provider service agents can interact directly to negotiate on behalf of their principals
6	Consumer and provider agent-mediated with referrals	Binding, selection, QoS monitoring, consumer-provider collaborations and negotiations, and consumer-only and provider-only collaborations and negotiations	Both consumers and providers use intermediary agents who can also both collaborate with other peer agents. Provider service agents can collaborate to dynamically provide capabilities that otherwise they would not

Table 1. Summary of Web services consumer-to-provider interaction styles.

The agent-mediated interaction styles use an agent mediator to automate some of the design-time tasks performed by the human designer in the direct interaction case. We differentiate three primary types of mediations: consumer-only mediation, provider-only mediation, and two-sided (consumer and provider) mediations. For each style, we further distinguish the case where the mediation is achieved solo from the case where the meditation is achieved using a group of other peer agent-mediators—forming a referral chain.

Figure 2 illustrates the consumer-only mediated interaction style. The consumer uses a service agent to mediate its interactions with service providers and implementations. The main mediation tasks performed by the consumer service agents include:

- Determining trustworthiness of service provider and implementation.
- Selecting service implementations.
- Binding service implementations.

The consumer service agents can share knowledge and opinions about their interactions with selected service providers and the providers' service implementations. Sharing knowledge and opinions help the community of consumer service agents improve their future tasks [14].

In contrast with the consumer agent-mediated interaction style, we can also place the agent on the provider. Figure 3 illustrates this case. A key advantage of this approach is that the provider's agent can help the provider learn the consumers' needs and thereby dynamically improve its service implementations.

We augment the consumer-only mediation interaction style by allowing the consumer service agents to directly communicate. Figure 4 illustrates this case. By directly

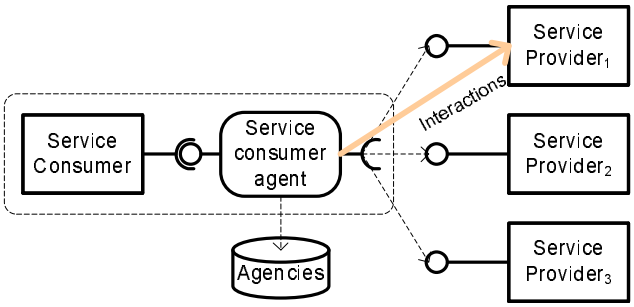

Figure 2. IS2: Consumer-to-provider intermediary service agent interaction style. Consumers use intermediary service agents to automate some of their design-time and runtime tasks and decisions. Agents consult agencies and service registries to find providers and share data.

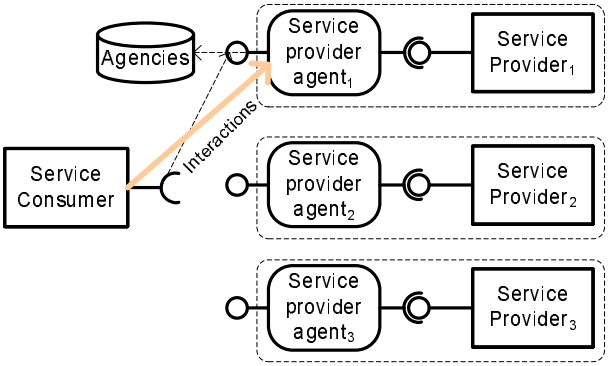

Figure 3. IS3: Provider intermediary service agents. The agents help the providers to dynamically improve their services, e.g., use agents to learn consumer needs to improve service implementation's QoS.

keeping a list of other consumer service agents, the agents create peer groups for collaboration. This approach naturally leads to referrals [19]. Each consumer service agent can advertise its expertise and offer advice to other consumer service agents. For example, a consumer service agent that has discovered an excellent service implementation that consistently satisfies its quality preferences, can advertise its historical quality usage experiences with that implementation as well as endorsing the service provider. Other consumer service agents use the information to make their service implementation selection decisions. This approach differs from using collective data for quality reputation calculation, since in this case the historical quality data being considered is from one consumer with a particular preference history.

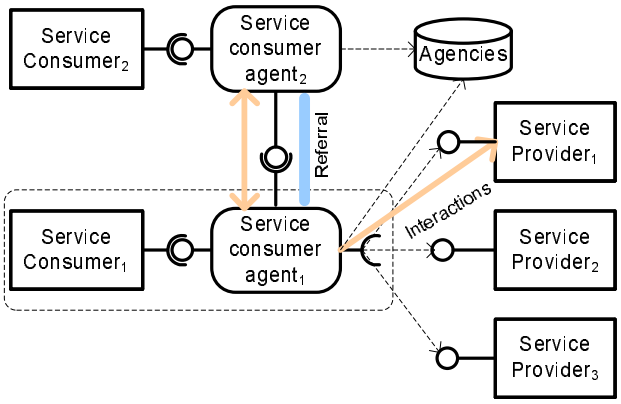

Figure 4. IS4: Consumer-to-provider intermediary service agent interaction style with referrals. The consumer service agents keep a list of other collaborator consumer agents which it uses to enhance its abilities.

The final kind of agent-mediated interaction style places intermediaries with both the service consumer and provider. Figure 5 showcases this interaction style. The addition of mediating agents on the provider enables dynamic interactions. For instance, the consumer and provider agents could directly negotiate agreements on quality levels. Elfatatry and Layzell [8] give an overview of negotiating in service-oriented environments that fits into our approach. Dynamic price negotiation, using auction mechanisms, is another possibility. Further, the provider agents could passively learn the shared quality opinions collected by consumer service agents and dynamically improve their service implementations—of course, we are assuming that the provider agents are allowed read-only access to the shared opinions.

We can extend consumer and provider service agents to accommodate referrals as we did for the consumer mediation interaction style. An advantage of adding referrals to provider service agents is to assign implicit trust (or endorsements) of one provider to another non-competing provider, perhaps one exposing complementary services. Also, providers could collaborate to dynamically merge services; thereby, offering a combined service that exposes service qualities that the service providers would not be able to achieve individually. The providers would then agree and negotiate on how to share the revenues from the combined service.

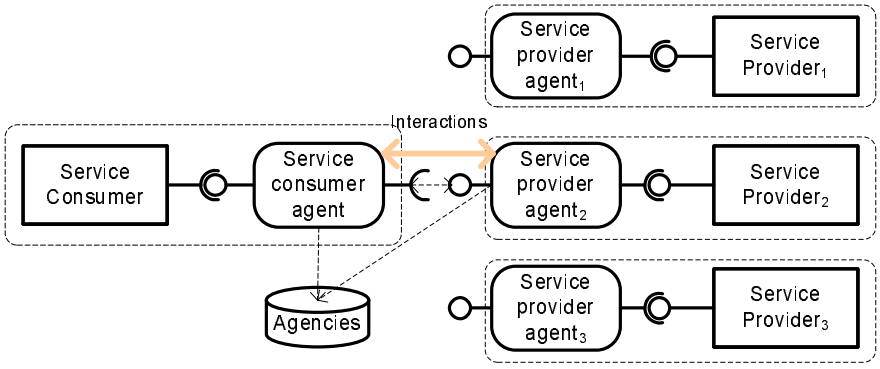

Figure 5. IS5: Consumer and provider intermediary service agents interaction style. Consumers and providers use intermediary service agents. The agents interact with each other on behalf of their principals, e.g., negotiate QoS levels guarantees.

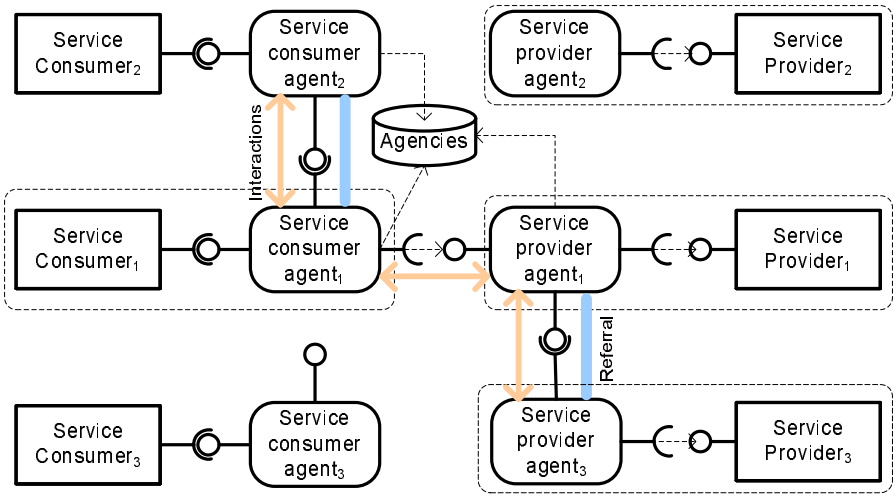

Figure 6. IS6: Consumer and provider intermediary service agents interaction style with referrals. The agents keep lists of peer collaborator agents.

Figure 7 summarizes the interaction styles that we presented showing how they build on each other.

4. Conceptual Evaluation

Architectural concepts such as the interaction styles that we propose are often not easy to evaluate empirically. The styles that we propose combine intuitions gleaned from years of research into distributed software by us and by others. It is still instructive to evaluate them, however, if only to better understand the circumstances in which they are helpful. The best evaluation for this purpose is by considering some important usage scenarios where the styles would help solve some important problems. These problems involve challenges in autonomic computing, creating ecosystem of services, dynamically improving service capabilities, and improving services with referrals, respectively.

4.1. Autonomic Computing

With the addition of software agents to service-oriented architectures (Table 1: IS2), we are able to automate various aspects between the interacting parties. Specifically, agents enable autonomic interactions for SOA in the following ways:

Service trust. SOA success depends on dynamically establishing trust between the interacting parties. Consumer

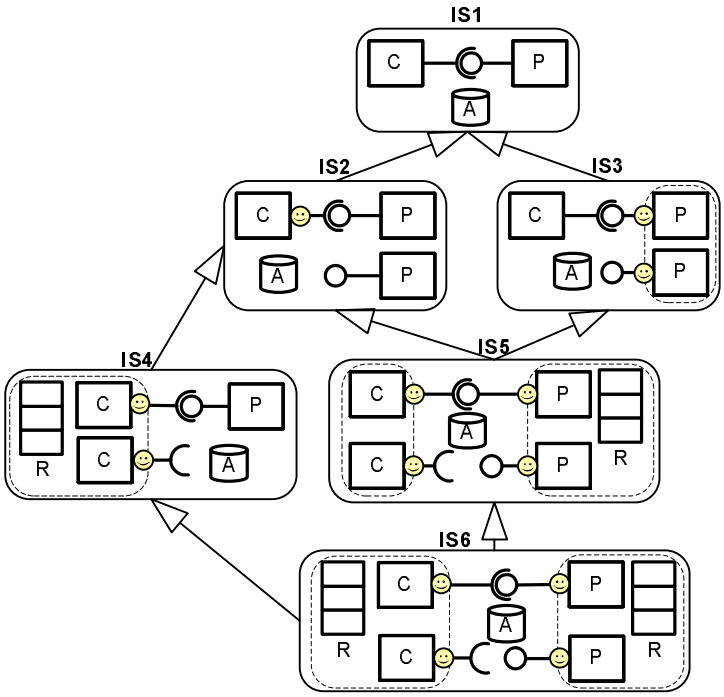

Figure 7. IS1–6: Summary of Web services interaction styles, showing how they build on each other.

service agent mediators yield a general framework to allow consumers to better determine service trust.

Service selection. The selection occurs at runtime and depends on the service's trust level, which itself partly depends on the service's nonfunctional characteristics. How can service selection be automated? Our framework provides an initial answer to this question. By attaching software agents to the consumers of services, we are able to automate service selection in a principled manner. Our approach, therefore, automates the selection interactions of service consumers to service providers.

Service binding. Our service agents enable the following kinds of bindings:

- *Late and dynamic binding*, in which the service implementation, after selection, is bound at runtime. The consumer-to-provider binding is delegated to the service agents.
- *Lazy binding*, which refers to when the service implementation is bound at the time of first need. A lazy interaction allows the consumer to only bind to the service when it is needed. This interaction style can minimize cost of binding, since if a service is never needed during a particular run it is never bound. The possible cost reduction is the key difference, since lazy binding avoids binding altogether whereas late binding merely delays it.
- *Rebinding*, wherein using policies our service agents can capture the conditions necessitating a rebind. For instance, when an error occurs during service usage, the consumer's policy may be to rebind. However, after a number of retries, the consumer's policy may be to engage in a reselection, potentially selecting an alternative service implementation better matching the consumer's preferences.

4.2. Monitoring Services

Our service agents are intermediaries between service consumers and service providers. As such, they can monitor the usage of services. With the help of shared conceptualization for services and QoS, the agents can capture quality data and share them via common agencies. In so doing, the agents enable service consumers to indirectly interact. The consumers share objective and subjective opinions on their selected services. This sharing of quality data enables QoS-based service trust models used by the consumers. Similarly, service agents added to providers (Table 1: IS3 and IS4) could learn the needs of consumers and how their service implementations are performing and, therefore, poten-

tially dynamically improve their service implementations' qualities.

4.3. Dynamic Service Improvements

The addition of intermediary service agents can enable dynamic and selective improvements of the service implementations and consequently the consumer's interactions with the implementations. The intermediary agent filters all service requests and responses and thus can be programmed to add layers of functionality that the original service did not initially expose. Examples of dynamic capability improvements of services are:

Security. The agents are programmed to secure the interactions between the consumers and the providers. For instance, for service implementations not supporting the WS-Security specification [2] a service agent could be programmed to interpose itself in the consumer-to-service interaction and implement the specification on their behalf.

Transactions. Adding transactions to services is usually thought of as the responsibility of the provider. However, what if the selected service does not implement the necessary transaction protocols but remains the best alternative of all available services—maybe due to other highly desirable qualities? In this case, the service agents could be programmed to layer transactional capabilities to the consumer-to-service interaction, e.g., by implementing suitable specifications, such as WS-AtomicTransaction [5].

Negotiations. With the addition of service agents on both consumers and providers (Table 1: IS5 and IS6), we could have the agents engage in dynamic negotiations to facilitate better matches of their policy needs. This is in contrast to approaches of passively relying on opinions of past interactions to make service selection decisions. Instead, with negotiations, we would allow consumers and providers to match actively and dynamically.

4.4. Service Referrals

Our multiagent architecture enables richer interactions among service consumers. For instance, each consumer service agent could keep a set of other collaborating consumer service agents (Table 1: IS4). These collaborator agents could then communicate directly, sharing their experiences with each other, and bypassing the common agencies. Such referral-style interactions could especially be interesting in the context of agents forming smaller communities of experts and novices searching for service implementations meeting certain specific quality criteria [19, 20].

5. Related Work

We divide related work into the following categories: (1) the pattern literature, in particular, software architecture patterns and software design patterns; and (2) the multiagent systems (MAS) literature and the growing literature on SOA—particularly works merging these two worlds.

5.1. Patterns

Pattern languages for various aspects of software systems from requirements, to analysis, to design, and software management are documented in [3, 10, 4]. Early work by Garlan and Shaw [12] motivated the architectural aspects of software but not necessarily in terms of patterns. More recently, Clements et al. [6] developed a thorough system of documenting software architectures. While none of these works specifically discuss interactions for Web services, our definition of Web services interaction styles is directly inspired from Garlan and Shaw's definition of software architecture styles and the pattern systems and languages that they discuss form the basis of our effort.

Another important related work is Fielding and Taylors conceptualization of the current Web architecture as Representational State Transfer (REST) [9]. REST can be thought of as a simple architectural style that treats a Web interaction as the execution of a (finite) state machine, whose transitions occur as the participating agents select links. In essence, REST provides a simple form of session management corresponding to the evolution of a shared state. REST is fundamental and supports our approach. However, our interaction styles layer interesting structures on top of REST that are geared for different uses involving Web services.

Our approach is founded on the basic idea that Web services are dynamic Web resources. Not only do they have a dynamically changing or temporal characteristic that differentiates them from static resources but in light of SOAs, this temporal characteristic reflected in evolving ratings can and should be modeled by providers and consumers alike.

Thus not only does representational state transfer take place when you interact with a given service, but many threads of state transfer take place—one for each of the key functions that we delineated, including modeling their qualities, sharing information about them, and consumer and provider centric mediation. In effect, our approach captures the multiple dimensions along with interactions need to be studied.

5.2. MAS and SOA

The use of multiagent systems and middle agents to facilitate Web usage is discussed in [18, 7]. While not focusing specifically on Web services, these works lay the

groundwork for modeling the Web as a system of interacting agents.

More recently, the DAML-S coalition [1] try to describe a complete ontology for making Web services usable by software agents. Their ontology is now called OWL-S, the Web Ontology Language for Services. Early results for dynamically discovering services using agents and OWL-S is presented in [17]. Our previous work [14] tries to complement the OWL-S efforts by adopting a similar agent architecture and focusing on nonfunctional aspects of services.

6. Discussion

As SOAs come about, it is becoming clear that the resulting ecosystem of service consumers and providers is leading to the creation of a service marketplace. Understanding the various interactions between service consumers and providers not only enables us to better understand the dynamism of the resulting marketplace, but also yields an understanding of the higher-level resulting business solutions that can be created using Web services. Using a multiagent framework that overlays Web services, we enable the automation of consumer and provider interactions. To better grasp the various possible interactions between and among service consumers and providers, we proposed a classification of interactions styles.

Implementing some of the proposed interactions is a challenge and will necessitate future research, such as into referrals among collaborating service consumers and negotiations between consumers and providers to dynamically improve service capabilities. For instance, providers specializing in security or specific domain capabilities, e.g., searching or mining content, can have their agents expose these functionalities to other service providers and therefore allow them to exploit these capabilities and enrich their own services to satisfy a consumer need. These specialized services could be advertised to other service provider agents for that explicit purpose, similar to how in the finance industry there exist credit processing and insurance services that local banks interact with in order to help them customize and better offer direct financial services to their customers.

An area for future research is in the creation of a complete pattern language for service interactions. The above classification is an initial step toward such a language. Such a language would provide service architects and designers prebuilt high-level business solution patterns.

7. Acknowledgments

We thank Paul Maglio of IBM's Almaden Research Center and Nirmit Desai of NC State University for useful discussions, and the anonymous reviewers for their comments.

References

[1] A. Ankolekar et al. DAML-S: Semantic Markup for Web Services. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, pages 411–430, July 2001.

[2] B. Atkinson et al. Web Services Security (WS-Security), Apr. 2002. Specification.

[3] P. Bramble, A. Cockburn, A. Pols, and S. Adolph. *Patterns for Effective Use Cases*. Addison-Wesley, 2002.

[4] F. Buschmann et al. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. John Wiley & Sons, Reading, MA, 1996.

[5] L. F. Cabrera et al. Web Services Atomic Transaction (WS-AtomicTransaction), Sept. 2003. Specification.

[6] P. Clements et al. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, Boston, 2003.

[7] K. Decker, K. Sycara, and M. Williamson. Middle-Agents for the Internet. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, Nagoya, 1997.

[8] A. Elfatatry and P. Layzell. Negotiating in Service-Oriented Environments. *Communications of the ACM*, 47(8):103–108, Aug. 2004.

[9] R. T. Fielding and R. N. Taylor. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, May 2002.

[10] M. Fowler. *Analysis Patterns: Reusable Object Models*. Addison-Wesley, Boston, 1996.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[12] D. Garlan and M. Shaw. An Introduction to Software Architecture. Technical Report CMU-CS-94-166, Carnegie Mellon University, Pittsburgh, PA, Jan. 1994.

[13] E. M. Maximilien and M. P. Singh. A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing*, 8(5):84–93, Sept. 2004.

[14] E. M. Maximilien and M. P. Singh. Toward Autonomic Web Services Trust and Selection. In *Proceedings of 2nd International Conference on Service Oriented Computing (ICSOC)*, pages 212–221, New York, Nov. 2004. ACM Press.

[15] D. C. Schmidt, R. E. Johnson, and M. Fayad. Software Patterns. *Communications of the ACM*, 39(10), Oct. 1996.

[16] M. P. Singh and M. N. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents*. John Wiley & Sons, Chichester, UK, 2005.

[17] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated Discovery, Interaction, and Composition of Semantic Web Services. *Journal on Web Semantics*, 1(1):27–46, Sept. 2003.

[18] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, Mar. 1992.

[19] P. Yolum. *Properties of Referral Networks: Emergence of Authority and Trust*. Ph.D. thesis, North Carolina State University, Raleigh, 2003.

[20] B. Yu. *Emergence and Evolution of Agent-based Referral Networks*. Ph.D. thesis, North Carolina State University, Raleigh, 2001.