Service Registry: A Key Piece for Enhancing Reuse in SOA

Juan Pablo García-González[1], Verónica Gacitúa-Décar[2], Dr. Claus Pahl[3]

[1] DATCO,Chile S.A.
[2,3] Software Engineering Research Centre, Dublin City University & Lero, Irish

Abstract

One of the promises of adopting a service-oriented approach in organizations is the potential cost savings that result from the reuse of existing services. A service registry is one of the fundamental pieces of service-oriented architecture (SOA) for achieving reuse. It refers to a place in which service providers can impart information about their offered services and potential clients can search for services. In this article, we provide advice for implementing an enterprise-wide service registry. We also discuss open issues in industry and academia that affect the management of service- repository information.

1. Introduction

The reuse of services greatly depends on the ability to describe and publish the offered functionality of the services to potential consumers (clients). A service registry allows you to organize information about services and provide facilities to publish and discover services.[1] Universal Description Discovery and Integration (UDDI) and the Web Services Description Language (WSDL)—together with SOAP— are standards for describing services and their providers, as well as how services can be consumed:

- **WSDL** [2] provides a model and XML format for describing what a Web service offers. A service description in WSDL separates abstract-service functionality from details such as how and where the service is offered. While the abstract-service description includes *types* and an abstract *interface*, concrete details include *bindings*, a *service* element that includes all available implementations of the abstract *interface* at *endpoints*.
- **UDDI**[3], [4] provides an infrastructure that supports the description, publication, and discovery of service providers; the services that they offer; and the technical details for accessing those services. A core aspect of UDDI is how it organizes information about services and the providers of services. Information entities (UDDI data) are organized in a data model and stored in a UDDI service registry. *Inquiring* (search and lookup entries) and *publication* (publish, delete, and update registry–related information) are core APIs.

Figure 1 illustrates some relationships between a WSDL service description and information that is stored in a UDDI service registry.

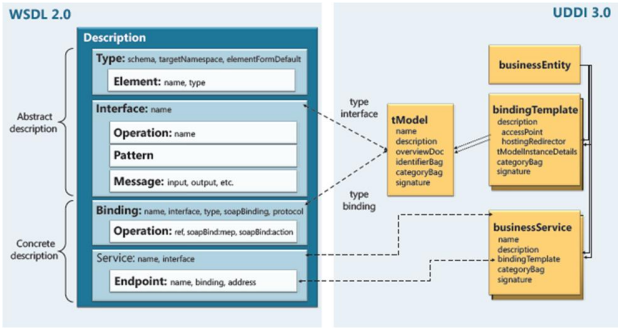

Figure 1. Relationships between WSDL and UDDI

Originally, UDDI was conceived to cover both publicly exposed services and services that were available within an organization. Currently, most existing implementations

are internal to organizations. Service publication, discovery, and (finally) *reuse of services* is more complicated in an inter-organizational scenario; for example, additional legal and commercial agreements are often needed among parties.

Dedicated (public) UDDI service registries were criticized for their limitations (among other reasons) during service inquiry/ discovery. Recently, however, Web search engines—which could be crawling publicly available WSDL documents—have raised promising expectations for discovering publicly available services.[5]

2. Designing an Enterprise Service Repository

This section proposes some design guidelines to develop an enhanced enterprise service repository. The focus is on improving the reuse of services over time in different IT projects. The aim is to increase service visibility to domain experts (often, this refers to a business-analyst role) and enhance service descriptions with practical information for architects. Business analysts, who have a less technical background but strong knowledge of the business domain, are frequently the early designers of new initiatives for incorporating or modifying the software support at companies; they play a key role with regard to the reuse of services.

2.1 Enterprise Services

Enterprise service–based solutions involve different types of service. Following the separation of concerns that is addressed by the *service-virtualization* pattern, [6] services can act as an intermediate layer between the client and provider applications of the services. The virtualization pattern focuses on the abstraction of technical details—such as service-endpoint location, policy enforcement, service versioning, and dynamic service-management information from service consumers— which access an *intermediate* service level. Technical concerns are managed at an *implementation* level, at which the actual business logic is implemented.

Based on SOA initiatives in several companies, we can identify three types of service:

- A business service (BS), which client applications use for accessing the functionality that is implemented in provider applications.
- An application service (AS), which can be consumed by a BS to access the functionality of the provider applications.
- A business-service extension (BSE), which can be consumed by a BS to operate on different AS responses and consolidate a single answer that is sent to a BS. In turn, the BS delivers the consolidated response to the client application. The aggregator pattern [7] is core to the design of a BSE.

Figure 2 illustrates the main static relations among elements of an enterprise service–based solution, as well as their relationship to elements from the virtualization pattern. A service registry organizes the description of the three different types of service and their relationships. Client and provider applications interchange messages that are mediated by BS, BSE, and AS. The service registry manages (at the configuration level) the information that relates the different types of service. The information is persisted in a service repository and used at runtime by a BS to answer client requests.

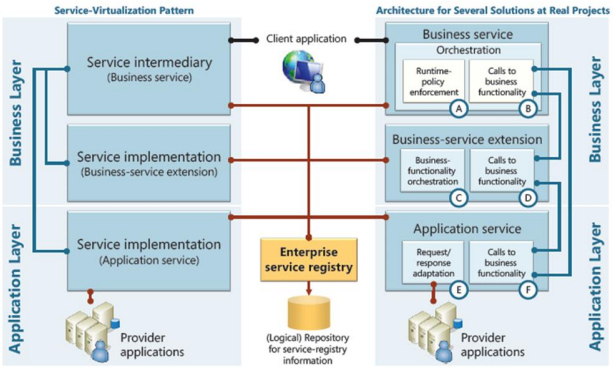

Figure 2. Service-based architecture and its relation to virtualization-pattern roles

2.2 Example 1

Let us consider a simplified BS that is used for calculating the total sales that are related to the *life-insurance* and *group-insurance* products of an insurance company. The total sales that are associated with *life-insurance* products

are obtained from a *life-insurance* legacy application. Analogously, the total sales that are associated with *group-insurance* products are obtained from the *group-insurance* legacy application. Each legacy application exposes an AS (*lifeInsurance* and *groupInsurance*, respectively) that provides the total sales for each type of insurance product. A BS receives requests from clients who are asking for the total sales; afterwards, it calls the service registry that has the information that is required to enforce specific policies on messages and dependencies to an AS and a BSE.

A BSE operates on the answers of an AS and provides a single answer to the BS that contains the total sales of the company. The BS, in turn, delivers this response to the client application. Figure 3 illustrates the described interaction.

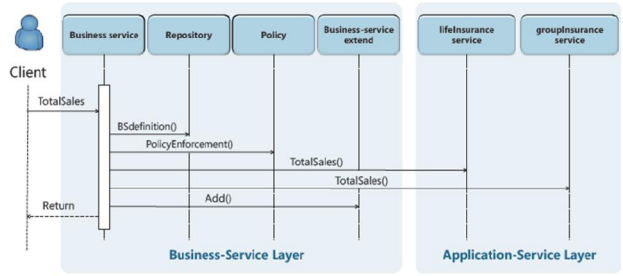

Figure 3. Main interaction among elements of the service-based solution from Example 1

2.3 Enterprise Service Registry

The enterprise service registry (ESR) is a core element that organizes service information and supports the interaction among enterprise applications that provide and consume services.

Basic functionalities of an UDDI-based ESR can be enhanced by using:

- Service-dependencies management.
- Runtime-policy enforcement.
- Service versioning.
- Service-history data (logs) management.

A service repository persists the information and documentation that are logically managed by the service registry. Figure 4 illustrates the main information that is organized in an ESR, persisted in a service repository, and provided to end users through a Web-based user interface.

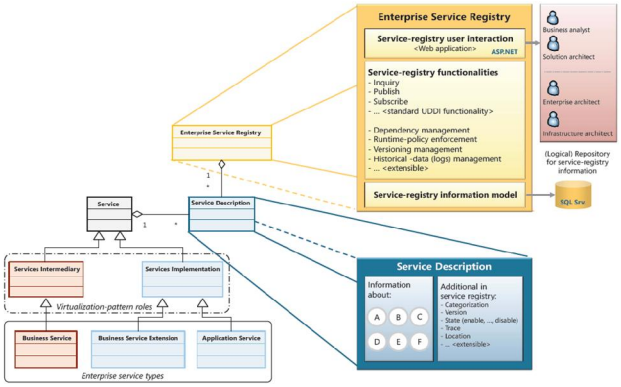

Figure 4. Enterprise service registry (ESR)

2.4 Services Descriptions

Services descriptions are core to the service registry. They determine how services can be discovered and subsequently reused:

- A BS is described at a high level—often, via textual descriptions in natural language and examples that facilitate understanding by business analysts.
- A BSE and an AS contain more technical details. A BS is implemented by at least one AS and also might involve a BSE. To associate a BS to one or more AS(s) and/or BSE(s), a dependency mapping is created and managed by the service registry.

Table 1 describes in more detail the information that is managed by the service registry:

- The main attributes that describe a BS are shown at the beginning of Table 1. A BSE and an AS share attributes (see middle of Table 1). The end of the table describes binding information that relates a BS to a BSE and an AS.
- Information that describes services and is independent of any registry implementation is shown in the **Service information** column, while information that is managed by the service

registry is shown in the **Registry information** column. If information in the **Service information** and **Registry information** columns is the same, an X appears in the **Replicated information** column.

- The remaining columns indicate information that is relevant to different roles. [8] Business analysts and solution architects manage information about business and technical concerns, respectively. Both roles work at a project or business-unit scale. Enterprise and infrastructure architects manage service information from a global (enterprise-wide) perspective. While enterprise architects might be interested in managing (for instance) service versions, infrastructure architects care about providing the required infrastructure support to keep services running with the adequate quality of service (QoS), as defined in service-level agreements (SLAs).

	Information			Role			
Business service (BS)	**Service information**	**Registry information**	**Replicated information**	**Business analyst**	**Infrastructure architect**	**Solution architect**	**Enterprise architect**
Service ID: Key used to access the service from client applications.		X		X		X	
Service name	X	X	X	X			
Service description: Textual description explaining what the service does and some other business related information such as what business objective the service addresses and if some business rules apply when it is utilized.		X		X			X
Input messages: Textual reference for input messages required to execute the service		X		X		X	X
Output messages: Textual reference for output messages generated after the execution of the service. Error messages might also be specified.		X		X		X	X
Category: Category(ies) to which the service belong(s). For example, one categorization schema used to classify business services was defined based on organizational business units.		X		X			X
Application service (AS) and business-service extension (BSE)	**Service information**	**Registry information**	**Replicated information**	**Business analyst**	**Infrastructure architect**	**Solution architect**	**Enterprise architect**
Service name	X	X	X	X		X	X
Service description: Textual description—often, together with a link to a WSDL file.	X	X	X	X		X	
Binding information: Technical information to access the service and subsequently execute it.	X	X			X	X	
Operation: Reference to functionality being implemented by the service.	X	X				X	X
Policies: Listing of runtime policies applicable to the service and link to associated documentation. Policies might include timeout, bandwidth, and debugging information (among others).		X			X	X	X
Category: Category(ies) to which the service belong(s). For example, one categorization schemas used to classify application services was defined by the technology implementing it. Different categories of SLA might also be associated.		X			X	X	X
State: Mainly, describes if a service is active or inactive. Intermediate states might also exist.		X			X	X	X
Message in: parameters	X	X				X	
Message out: response	X	X				X	
Binding: BS to AS and BSE	**Service information**	**Registry information**	**Replicated information**	**Business analyst**	**Infrastructure architect**	**Solution architect**	**Enterprise architect**
Business-service name	X	X	X	X		X	X
Application-service and business service–extension names	X	X	X	X		X	X
Message in: parameters MAP (dependencies)		X		X		X	X
Message out: response MAP (dependencies)		X		X		X	X

Table 1. Main service information at the ESR

2.5 Using the Enterprise Service Registry to Improve Reuse of Services

Based on our experience in a range of projects, providing simple descriptions about a BS, facilitating its access, and managing services dependencies have been key to improve reuse. For this purpose, an ESR was a core element.

- Business analysts who trigger new requirements for software support can improve their communication with solution and enterprise architects by referring to a BS that is described in the ESR. Based on the descriptions of the BS and its dependencies to an AS and a BSE, domain experts become aware of available functionality at back-end applications. From our experience, this has facilitated a shift from requirements that are specified in a vague manner to initial solution blueprints that comprise orchestrated services (created by business analysts). Long meetings between architects and business analysts can be reduced to short meetings or even telephone calls that refer only to information at the service registry.
- Software architects can refine orchestrations that are depicted in the initial blueprints that are made by business analysts. Subsequently, they can agree with enterprise and infrastructure architects on service versions and infrastructure support. Again, information at the ESR was central during the agreement.
- Information about service dependencies helped infrastructure architects to analyze the impact of binding new consumers to application services. This is critical for maintaining SLAs.
- Runtime policy–enforcement configurations at the service registry allowed specialized treatment for different client-application requests that were associated with a single BS—for example, applying particular validations with regard to formatting, security, and parameterization.
- In the case of new requirements triggering modifications to existing services:
- Often, extensions or modifications involved changes only at the BSE level.
- If an AS or BSE was modified and new versions were deployed, the version of the associated BS remained unchanged. (Service versioning is discussed in more detail in the “Impact of Service Versioning on Service Registries” section.)
- Only incompatible changes that modify the business functionality could trigger new BS versions (in general, a BS is designed with forward compatibility in mind).

Among the lessons that were learned from different projects, we can emphasize the following:

- Decoupling of a BS from an actual implementation (by using AS(es) and a BSE) is an effective way of keeping domain experts separated from technical information, which facilitates service discovery at the business level.
- BS discovery support is key to enable the reuse of services beyond a single solution or project—allowing their use across projects and at an enterprise-wide scale.
- In practice, when only an AS is presented to domain experts, it remains almost untouched; that is, it is rarely reused in further developments.
- Even when new requirements involved modifications to existing service solutions, the reuse of a BS has still been strong. This was facilitated by addressing the required modifications at the BSE level.
- During legacy-application migration, client applications kept consuming the same BS. Changes mostly occurred at the AS level. At the ESR, AS descriptions and service dependencies were updated. New projects could reuse a BS independently when a migration had occurred.

3. Open Issues in Industry and Academia

This section discusses a number of observations in industry and academia with regard to enhanced service descriptions, organization of service information in a service registry, and the role of such a registry to enhance the reuse of services.

3.1 Strategies for Organizing and Finding Services in Registries

If service information in an enterprise service registry is difficult to distinguish because of inadequate organization or ineffective search mechanisms, the value of that registry is reduced.

Services *categorization* can help to distinguish services and classify them according to one or more categories. UDDI registries support this through the tModel. The categorization schemas of UDDI refer to taxonomic classifications. *Taxonomies* organize concepts in a hierarchical structure; multiple taxonomies can apply to a single UDDI entity. Standard classification schemas are suggested, such as the United Nations Standard Products and Services Code (UNSPSC [9]); however, other standards or internally created taxonomies can also be used. The UDDI Inquiry API supports different forms of query, such as browse pattern, drill-down pattern, and invocation pattern. *Queries* can refer directly to services, as well as to *service categories*.

Similarly to a Web search engine, the browse pattern allows one to find registry elements by matching keywords. Although this mechanism automates part of a service search, the results are limited to the coding system's value set and direct value matching. Services whose description includes similar or related concepts, but different syntax, cannot be retrieved by using this approach. Also, during use of different categorization schemas, the management of overlapping categories can become expensive. [10] Taxonomy maintenance is an added load that must be considered during the implementation of a service registry. Classification schemas that are not updated can affect the quality of the discovery results. [11]

The semantic research community has proposed alternatives to enrich service descriptions semantically and enhance classification schemas in services registries. Basic taxonomies can be enriched or replaced by ontologies. Ontologies structure concepts within a domain and define their *meaning*. Axioms constrain possible interpretations of concepts and reasoning mechanisms that support inferences from existing data.

According to Küster *et al.*, [12] although semantic enrichment of services descriptions can improve service discovery, several issues still must be addressed, such as reducing the computational cost of reasoning, maintaining the ontologies, and refining search results to improve effectiveness.

3.2 Impact of Service Versioning on Service Registries

When a service has been implemented, changes can occur—from the implementation itself to parts of the service description in a service repository. Changes might aim to improve reuse:

- Implemented services that follow a *bottom-up* approach [13] often fulfill particular project requirements within a domain. When any of these services becomes a candidate for reuse in a different context, it usually requires modifications or extensions.
- Analogously, services that follow a *top-down* approach often must be changed (specialized) to fit in particular contexts.
- Different versioning strategies address different requirements. A single solution is not likely to be satisfactory for all situations. WSDL and UDDI do not define guidelines for versioning services. Some authors have proposed strategies for service versioning; most of them relate to *backward* and *forward* compatibility: [14]
- A *backward-compatible* version refers to the ability to support consumers of older versions of a service.
- A *forward-compatible* version refers to the ability to adapt to unknown future requests that are intended for newer versions of the service. This type of compatibility involves not only a service-*versioning* strategy, but also a service-*design* strategy that is related to *changeability*.

Often, new service versions are replications of a previous version that have additional or modified elements. New versions are named differently (by using some naming convention), and their description is stored in the registry as a new entry. Juric *et al.* [15] propose extensions to WSDL and UDDI for service versioning. The approach addresses run-time and development-time versioning. Efficiency at the code level is addressed by allowing multiple versions of a service to refer to the same codebase. Additionally, notifications about new and deprecated versions are communicated to consumers. Traceability support is provided to track changes. This academic research promotes the reuse of services and keeps the complexity of a service registry manageable.

3.3 Service-Usage Information for Enhancing Service Description and Discovery

The history of service usage can be an interesting source of information—not only to re-create the actual behavior, [16] but also for service discovery. Stored service usage–history (logs) can help to categorize services according to the user or how services have behaved over time. Let us consider a service description that indicates a specific performance level in its contract; however, the actual measured performance in a given timeframe (extracted from logs) is lower. This information could be used during service discovery; a service that had lower-than-expected performance levels would be discarded from the search.

Statistically extracted information about how services behave against historic interactions can help to build less biased rankings and make service discovery more precise; however, an infrastructure for the constant monitoring of services and storing of the history information must be provided. Based on the service history, probabilities can be assigned to quantify uncertainty. Clark *et al.* [17] consider uncertainty with regard to the configuration of a service-based system, the rate parameters of system components, and the duration of events. An uncertainty model is used to predict system performance under increased demand. This type of analysis is fundamental when one is dimensioning the service support infrastructure. Historical data about individual services helps to predict the performance of an entire system.

Offer and demand in an inter-organizational scenario are subject to how much parties trust one another. “Trust in others” is one of several criteria for assigning reputation—witness reputation [18] —to publicly available services. If company X knows that a service is being used or was positively rated for company Y, whom X trusts, the reputation of that service would increase from the point of view of X. One associated problem is the eventual bias for positive ratings, unfair ratings, and the variations of quality between ratings. [19]

3.4 Sufficiency of WSDL Descriptions to Find Services for Composition Efficiently

Services are reused not only by client applications, but also by other services in a service composition. A service composition can provide a more coarse-grained

functionality and be closer to a business need. One problem when finding a service (useful in a service composition) is the need to verify if the services that are involved are able to "talk" to one another—that is, if the associated message-interchange protocol among them is *compatible*. A basic requirement for compatibility is deadlock-freeness. Moreover, the message syntax and semantics should be compatible.

Figure 5 illustrates a typical example of incompatibility at the protocol level between two parties. In the figure, a *Buyer* party offers a *buyProduct* service, and a *Seller* party offers a *sellProduct* service.

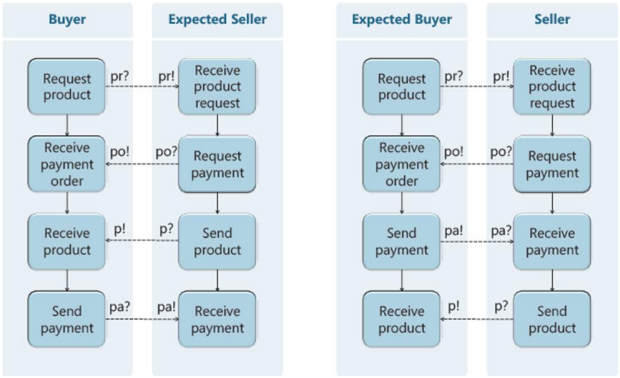

Figure 5. Example of incompatibility at protocol level between two parties

To automate a hypothetical sale process, the message-interchange protocol between *buyProduct* and *sellProduct* should be compatible. However, Figure 5 illustrates that the Seller expects a payment before sending the product, and the Buyer expects the product before sending the payment.

When more and more services are offered and advertised in repositories, there are more chances of satisfying a service demand by composing existing services. However, mediation at the protocol level might be required. Matchmaking conflicts at the message and/or conversation level(s) can be solved—to a certain degree— by a mediator component. [20], [21] However, verifying and solving compatibility among services at the behavioral level is expensive; it involves the (expensive) exploration of possible states of the services during interaction. To increase reuse here, we need efficient mechanisms for finding compatible services.

For instance, instead of directly publishing the behavior of a service in a repository, a provider can publish a "summarized description" of the expected behavior of all compatible services to service (compatibility refers to deadlock-freeness). The "summarized description" is called an "operating guideline [22]" and allows the hiding of implementation details, while exposing enough information to find compatible partners. Checking if a service can be composed with others is reduced to checking if a graph-based representation of the potential partner is a sub graph of the "operating guideline," which is less expensive than exploring all possible states of the services.

4. Conclusions

To improve the reuse of services at the enterprise level, architects must define a strategy for publishing and providing facilities to access services information. For this purpose, an enterprise service registry is a key piece. Information about services can be organized at the registry, and basic functionality can be enhanced—including, for instance, functionality for service versioning, management of service dependencies, and enforcement of runtime policy. In this article, we have provided some design guidelines for enhancing an enterprise service registry to improve the reuse of enterprise services. We have also discussed some open issues in industry and academia with regard to the design and implementation of service registries and associated aspects that are required to describe and organize services information.

Resources

1. Pijanowski, Keith. " Visibility and Control in a Service-Oriented Architecture." MSDN, May 2007.
2. W3C.org. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language.
3. UDDI.org. UDDI Version 3.0.2. *UDDI Spec Technical Committee Draft,* October 19, 2004.
4. Microsoft Corporation. " UDDI Specification Index Page." MSDN, July 2009.
5. Al-Masri, Eyhab, and Qusay H. Mahmoud. " Investigating Web Services on the World Wide Web." *WWW 2008: Web Engineering— Web Service Deployment Track*, April 21–25, 2008, 795–804.
6. Madrid, Chris. " SOA Realization Through Service Virtualization." *SOA Magazine*, September 3, 2007.

7. Hohpe, Gregor, and Bobby Woolf. *Enterprise Integration Patterns*. Boston: Addison-Wesley, 2004.
8. Cardinal, Mario. " The Hidden Roles of Software Architects: The Three Types of Architect." MSDN, March 2008.
9. UNSPSC.org. " UNSPSC Codeset." (Managed by the Electronic Commerce Code Management Association (ECCMA).) August 2007.
10. Klein, Michel. "Combining and Relating Ontologies: An Analysis of Problems and Solutions." Proceedings of Workshop on Ontologies and Information Sharing at 17th International Joint Conference on Artificial Intelligence (IJCAI-01), 2001, 53–62.
11. Hepp, Martin, Joerg Leukel, and Volker Schmitz. "A Quantitative Analysis of eClass, UNSPSC, eOTD, and RNTD: Content, Coverage, and Maintenance." Proceedings of IEEE International Conference on e-Business Engineering (ICEBE'05), 2005, 572–581.
12. Küster, Ulrich, Holger Lausen, and Birgitta König-Ries. "Evaluation of Semantic Service Discovery: A Survey and Directions for Future Research." *Emerging Web Services Technology*. Volume II, 2008, 41–58.
13. Erl, Thomas. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall Service-Oriented Computing Series from Thomas Erl, 2004.
14. Parrish, Allen, Brandon Dixon, and David Cordes. "A Conceptual Foundation for Component-Based Software Deployment." *Journal of Systems and Software*. Vol. 57, Issue 3, July 2001, 193–200.
15. Juric, Matjaz B., Ana Sasa, Bostjan Brumen, and Ivan Rozman. "WSDL and UDDI Extensions for Version Support in Web Services." *Journal of Systems and Software*. Vol. 82, Issue 8 (doi:10.1016/j.jss.2009.03.001), August 2009, 1326–1343.
16. van der Aalst, W. M. P., and A. J. M. M Weijters. "Process Mining: A Research Agenda." *Computers in Industry*. Vol. 53, Issue 3 (doi: 10.1016/j.compind.2003.10.001), April 2004, 231–244.
17. Clark, Allan, Stephen Gilmore, and Mirco Tribastone. "Quantitative Analysis of Web Services Using SRMC." *SFM*. Vol. 5569, 2009, 296–339.
18. Huynh, Trung Dong, Nicholas R. Jennings, and Nigel R. Shadbolt. "An Integrated Trust and Reputation Model for Open Multi-Agent Systems." *Journal of Autonomous Agents and Multi-Agent Systems*. Vol. 13, Issue 2, March 2006, 119–154.
19. Jøsang, Audun, Roslan Ismail, and Colin Boyd. "A Survey of Trust and Reputation Systems for Online Service Provision." *Decision Support Systems*. Vol. 43, Issue 2, March 2007, 618–644.
20. Dustdar, Schahram, and Wolfgang Schreiner. "A Survey on Web Services Composition." *International Journal of Web and Grid Services (IJWGS)*. Vol. 1, Issue 1, 2005, 1–30.
21. Li, Xitong, Yushun Fan, Jian Wang, Li Wang, and Feng Jiang. "A Pattern-Based Approach to Development of Service Mediators for Protocol Mediation." Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), February 2008, 137–146.
22. Kaschner, Kathrin, Peter Massuthe, and Karsten Wolf. " Symbolic Representation of Operating Guidelines for Services." *Petri Net Newsletter*. Vol. 72, April 2007, 21–28.

Juan Pablo García-González is a senior software developer and chief architect at DATCO Chile S.A. He has been involved in numerous software projects involving service-based solutions. In addition to their regular projects, he and his team are working on the creation of innovative service-centric mobile solutions for the banking industry.

Veronica Gacitua-Decar is a postgraduate researcher at the School of Computing, Dublin City University, and Lero - the Irish Software Engineering Research Centre. Her research is focused on the development of tools and methods for designing process-centric service architectures. Previously, she worked as an IT architect in a large mining company.

Dr. Claus Pahl is a senior lecturer at the School of Computing, Dublin City University, where he is leader of the Software and Systems Engineering group. He is also involved in Lero - the Irish Software Engineering Research Centre, as well as the Centre for Next Generation Location (CNGL).