Interoperability in Service-Based Communities

Toni Ruokolainen and Lea Kutvonen

Department of Computer Science,
P.O. Box 68 (Gustaf Hällströmin katu 2b),
FI-00014 University of Helsinki, Finland
Toni.Ruokolainen@cs.Helsinki.FI, Lea.Kutvonen@cs.Helsinki.FI

Abstract. Interoperability is a multifaceted problem caused by issues surpassing those of technological incompatibilities. The real interoperability challenges are stemming from various sources, such as organisational incompatibilities buried deeply into the structures of collaborating enterprises, architectural mismatches and defective assumptions about business application behaviour, or from the inherent properties of business collaboration models.

To achieve interoperability in enterprise computing environments, the aspects of interoperability must be identified and their properties analysed. This paper studies interoperability issues in enterprise computing environments. Enterprise computing environments under analysis are based on Service Oriented Computing paradigm and enhanced with necessary infrastructure facilities. Several classes of causes for interoperability problems are identified and the mechanisms for overcoming the problems in these classes are briefly discussed.

1 Introduction

Interoperability of business applications in enterprise computing environments is a challenging problem. Work addressed by such projects as INTEROP [15] and Athena [3] try to grasp the different angles of interoperability issues in enterprise applications and systems. Issues related to purely technological interoperability can nowadays be usually handled with use of a common distributed object computing platforms such as CORBA, or by exploiting service oriented architectures and Web Services.

Interoperability is nevertheless a multifaceted problem caused by issues surpassing those of technological incompatibilities. The real interoperability challenges are stemming from various sources, such as organisational incompatibilities buried deeply into the structures of collaborating enterprises, architectural mismatches and defective assumptions about business application behaviour, or from the inherent properties of business collaboration models.

The notion of interoperability has been left quite vague both in the academia and industry. However, for achieving interoperability in enterprise computing environments, the aspects of interoperability must be identified and their properties analysed. This paper studies interoperability issues in enterprise computing environments. Several classes of causes for interoperability problems are identified

C. Bussler et al. (Eds.): BPM 2005 Workshops, LNCS 3812, pp. 317–328, 2006.

and the mechanisms for overcoming interoperability problems in these classes are briefly discussed.

Section 2 discusses interoperability in general and describes different models of collaboration and typical means of achieving interoperability in these models. Section 3 introduces a taxonomy for interoperability in context of federated service communities. After introducing the taxonomy, methods and mechanism for establishing interoperation with respect to the identified aspects are briefly discussed in Sections 4, 5 and 6.

2 Interoperability Methods

Interoperability has been defined as the ability of two or more entities to communicate and operate together in a meaningful way, such that information gets exchanged between collaborating parties and it is used in a meaningful way despite differences in language, interface or operation environments [14, 18, 45]. Interoperability, or capability to collaborate, means effective capability of mutual communication of information. Interoperability covers technical, semantic and pragmatic interoperability. Technical interoperability means that messages can be transported from one application to another. Semantic interoperability means that the message content becomes understood in the same way by the senders and the receivers. This may mean both information representation or messaging sequences. Finally, the pragmatic interoperability captures the willingness of partners for the actions necessary for the collaboration. The willingness to participate involves both capability of performing a requested action, and policies dictating whether the potential action is preferable for the enterprise to be involved in.

Three different forms of model interoperability have been identified in [16]. The identified forms of interoperation, namely integrated, unified and federated interoperability, can also be identified from enterprise computing environments as different models for collaboration. These models of collaboration are distinguished from each other on the grounds of where information needed for achieving interoperability is found.

In the integrated model of collaboration the knowledge for ensuring interoperability is implicitly injected into different software components of the computation system. The integrated collaboration model can be furthermore classified into three different solution methods based on the deployment point of the interoperation knowledge. The solution methods are 1) tightly coupled integration, 2) software adaption, and 3) use of common computing environments (middleware).

In tightly coupled integration the knowledge needed for interoperation is weaved inside the business applications implicitly by software designers and implementors. This has been a very popular approach to achieve interoperability of intra- and inter-enterprise business applications in the past. The implicit assumptions about other components and operating environment however make this tightly coupled integration and interoperation model an impermanent

solution in general. Tight coupling is however very cost-effective and easy to implement if there is an absolute certainty that there will never be any changes in any part of the system.

In software adaption approach interoperation knowledge and functionality related to interoperation is partly isolated from the application components. Interoperation knowledge is injected into intermediary software components such as software adapters and wrappers. Adapters are used to mediate incompatibility of software entities [45] and they can perform mappings between data values and schema structures or even adapt the behaviour of services [5]. Wrappers are used for introducing completely new behaviour that is executed usually before or after the actual functionality [6].

Third form of integrated collaboration is based on use of a common computing environment. In this method the interoperability knowledge is located in computation and communication facilities of the system. A common computing environment provides a homogeneous technology and communication platform as well as computation model to be used in an enterprise computing environment. Interoperability is mediated between business applications and the computing platform via an intermediary language, an interface description language (IDL) [22]. Middleware platforms such as CORBA [28] or J2EE [41] and interface description languages have been used successfully to bridge technological differences between operation environments.

Second model of collaboration in enterprise computing environments is the unified model. In unified collaboration model a shared meta-information entity describes the functionality and responsibilities of each community participant. Two kinds of meta-information entities can be identified: standards and explicitly shared meta-information.

Interoperation has been achieved in traditional forms of industry through standardisation. However, in software engineering, standardisation of software entities such as components or even communication technology has not been as successful. First of all, software components and computing systems themselves are usually highly dynamic entities whereas standardisation processes are slow with respect to the advances in ICT technology.

Secondly, not even standardisation guarantees interoperation if either standards are too ambiguous or developers do not comply to the standards. Interoperability problems between implementations of the same CORBA service from different vendors were studied in [4]. The result was that the formal specification for OMG's CORBA Event Service middleware function was too ambiguous and underspecified for guaranteeing interoperable and substitutable implementations of the same standard. Interoperability problems stemming from different interpretations of the same standard or too loose standardisation are found also from Web Services technology [36].

Unified collaboration can be achieved by use of explicitly shared meta-information. Meta-information defined using an appropriate modeling language, such as UML, describe component functionality, their properties and interrelationships using computation platform independent notations. Typically the

meta-information, or a model is used for generating the actual business application components. As the components implemented by generative methods are based on the same platform independent conceptual model, interoperation between components generated by different vendors should be possible, given appropriate code generation tools. The most renown representative of this approach is OMG's Model-Driven Architecture initiative [11].

Federated collaboration means that no shared, native meta-information describing the operation of a collaboration is presupposed or needed. Each participant may have their own models describing their business services. To achieve interoperability a *shared meta-model* is exploited. Interoperation is established by negotiation mechanisms, model verification and monitoring of service behaviour with respect to the interoperability contract. Meta-information needed for ensuring interoperability must be explicitly available, especially during operation of service communities. Federated collaboration model needs additional infrastructure facilities for publication and management of meta-information, and for controlling and monitoring the communities.

The model of integrated application collaboration provides solutions for establishing technical interoperability. Heterogeneity in technical level is supported but usually there are a very strict bindings between the collaborating business applications and underlying computation and communication platform. Therefore heterogeneity in higher abstraction levels, such as service behaviour or information representation levels, is not usually tolerated. Integrated collaboration model does neither tolerate dynamism or autonomy of participants. As the information about interoperation prerequisites is hidden inside business application and infrastructure components, dynamic changes in the system can not be coped with.

Unified collaboration model provides support for both technical and semantic interoperability. Heterogeneity of computation and communication platforms is also supported, assuming that the meta-information is platform independent. Unified collaboration based on shared models is however inflexible due to the fact that although the design entities in the models are reusable, the actual service components are typically specialised for the specific architecture and use-case described in the model. Model evolution is effectively supported but real dynamism, that is awareness and adaption to runtime changes is not supported by the unified collaboration model, unless explicitly modeled in the meta-information.

Interoperability is an issue which can not be fully realised by homogenising the execution environment through distribution middleware, or by using unified meta-languages or some other means of mediation. Object and component interoperability even in a homogeneous computing platform is a multi-faceted issue with syntactic, semantic and behavioural aspects [44]. When considering enterprise computing systems with heterogeneous implementation platforms and autonomous participants, interoperability of software components becomes even more complicated, since these kind of computing environments are characterised by their heterogeneity (freedom of design), autonomy (freedom of action)

and dynamism (freedom of configuration) [38, 39]. Both integrated and unified collaboration models support at least partially technological heterogeneity in enterprise computing environments. They however fail to address the autonomy and dynamism aspects, and do not provide pragmatic interoperability.

To establish pragmatic interoperability needed in enterprise computing environments, the federated collaboration model should be used. Federated collaboration provides support for heterogeneity in technology, computation, communication and information levels via loose coupling of business applications and contract based co-operation. Negotiation mechanisms and monitoring facilities provide support for runtime dynamism and autonomy over service activity. Model evolution is also supported, as interoperability between business applications is achieved via shared meta-model and interoperability validation facilities. Service Oriented Computing approach [30] is especially suitable framework for federated collaboration model as it promotes use of self-descriptive, independent and composable software entities and loosely coupled collaborations based on the notion of contracts.

3 A Taxonomy of Service Interoperability

To establish and support an open model of interoperability, the different aspects of interoperation must be identified and analysed. The notion of interoperability must be separated into independent aspects, each aspect grasping a different need or viewpoint of enterprise computing. A clean separation of aspects is important because otherwise it would be very difficult to identify the requirements interoperability imposes on modeling concepts and infrastructure facilities.

In this section a taxonomy of interoperability aspects for service based communities using a federated collaboration model is presented. Interoperability is classified into different abstraction layers, each layer grasping more abstract interoperation concepts than the previous one. Classification is based on previous studies of interoperability (see for example [9, 44]) and on conceptualisation of enterprise computing environments made in web-Pilarcos project [21].

Interoperability in federated communities is divided into five abstraction levels: 1) technology, 2) service, 3) community, 4) organisation, and 5) business level. This division is based on identification of the subjects responsible for deciding if interoperation can be achieved. Each level is further divided into different aspects; the classification as a whole is illustrated in Figure 1.

At the technology level, technical interoperability must be achieved between communication and computation platforms. Interoperation is established by selecting and configuring appropriate middleware services and their parameters. When we consider only technical interoperability, that is the connectivity, communication and encoding related aspects, incompatibilities between languages, interfaces or operational environments can be solved quite efficiently. Methods and techniques like interface description languages [22, 27], adaptors [34, 46], wrappers [26], middleware [28,41] and middleware bridges [10] have quite success-

Fig. 1. Aspects of interoperability

fully been applied for enterprise integration. However, while providing the necessary means for collaboration, technology level interoperability and the methods for achieving it are only the basis of the "interoperability stack".

At the service level, both technical (compatibility between service signatures) and semantic interoperability (semantics and behaviour of services) between service end-points must be established. Service discovery mechanisms are used for this purpose and the decision making procedures are bilateral. Service level interoperability means capability of interoperation between electronic services with well-defined, self-descriptive interfaces.

Interoperation between distinct services does not guarantee that functionality of the whole system is consistent and flawless. Requirements and constraints for application interoperation are induced by the global properties of the community in question. For establishing community level interoperability the aspects of architectural properties, failure handling procedures and compensation processes, and non-functional aspects of communities must be addressed. Decision making at the community level is multi-lateral since the properties of all the participants must be taken into consideration. Negotiation mechanisms are used for populating communities with compatible services. Both technical (non-functional aspects) and semantic interoperability (failure handling and community architecture) is addressed at the community level.

Pragmatic interoperability is addressed at the organisation and business levels. Business rules and policies must be agreed upon at the organisation level. Organisation level interoperability deals with issues related to the needs of autonomic enterprises. Policies and business rules are business knowledge which must be explicitly represented if inter-organisational collaboration should be achieved. Policies are used to constraint community behaviour such that the common objective of the community can be achieved [40]. Business rules are declarative rules that constraint or define some aspects of business [12]. Both policies and business rules are organisational entities that are independent of community or service life-cycles; thus it is necessary to separate these aspects from the aspects related to community and service level interoperability.

Organisation level is the last level of abstraction that is embodied as explicitly available meta-information. Corporation business strategies and legislation con-

cerning for example geographic regions or business domains are typically available as implicit regulations and constraints at the lower levels. In the following sections we will discuss three most important levels of the presented taxonomy in more detail, namely the service level, community level and organisation level interoperability.

4 Service Level Interoperability

Interoperability at the service level is characterised by three aspects, namely syntactic, semantic and behavioural properties of service interfaces [43, 44]. An interface is an abstraction of application functionality which decouples the internal implementation details from the externally provided service. A service interface description provides definitions of the service syntax (interface signature and document structures), semantics and its behaviour.

Service level interoperability has been studied mainly among object oriented and component based approaches [8, 18, 43, 44]. Object oriented interoperability was first addressed in [18]. This work recognised that interoperability conflicts in object oriented platforms can not be solved by simple adaption or procedure parameters between heterogeneous objects with use of unifying type systems. It is the overall functionality and semantics of an object which is important [18].

Substitutability and compatibility of software entities can be considered as the most relevant concepts in this level. When considering syntactic and semantic aspects we are interested if two entities can be substituted by each other. The concept of compatibility is relevant only when behavioural aspects are taken into consideration.

Validation of syntactic interoperability, that is substitutability of syntactic structures, reduces to type matching. Type matching problem is about finding and defining bindings and transformations between the interface a client wants to use and the interface provided by a service [18]. Type matching problem in general is impossible, since identification of operation semantics and information contents used in the operations or attributes can not be fully automated. However, if two interface signatures are described using the same language (type system) or the interface descriptions can be unified, and only syntactic properties of services are considered, efficient type matching methods and algorithms can be used [17, 29].

When considering type matching in Web Services based environments, the notion of schema matching emerges (see for example [32, 33]). Schemas define document structures used for information descriptions. When considering Web Services based environments, XML-Schemas are used for describing document structures. The type system behind XML-Schema mixes both structural and name-based features [37]. This makes XML-Schema matching a bit complicated and the type system less elegant, since purely structural matching methods cannot be used.

Semantic aspect of service level interoperability is concerned with the meaning of service operations and documents. Matching of service interfaces based on

their operational semantics have been addressed for example in [47]. Operational semantics are usually attached to a service as operation-specific pre- and post-conditions (or effects). These conditions are definitions given in appropriate logic describing the assumptions and results of the operations.

Semantics are use also for attaching meaning for information contents exchanged between services. In tightly coupled and closed systems interpretation of semantics is implicitly coded into the applications, since the operational environment is known during development of the application. Exploiting explicit shared ontologies for description of operation and information semantics provides a more loosely coupled approach. Attaching semantics to service operations and messages for establishing interoperation of services sharing a common ontology is the approach taken for Semantic Web services [25, 31]

Attaching behavioural descriptions to interface signatures provides stricter guarantees of service interoperability. When only syntactic and semantic aspects are considered, we cannot clearly specify how the service should be used. If a formal specification of behaviour is attached to service interface, compatibility or equivalence of services behaviours can be verified using formal methods [7,46].

5 Community Level Interoperability

Interoperability at the community level must be guaranteed with respect to non-functional aspects, failure handling mechanisms and architectural properties. Interoperation is established by multi-lateral negotiations during community breeding process [20]. Interoperation at the community level is a mutual agreement between all the participants. Community level interoperability grasps rest of the semantic interoperability aspects in enterprise computing environment in addition to the semantic and behavioural aspects described at the service level.

Agreement of non-functional properties, such as quality of service, security, trust, location or availability is an important aspect in community level interoperation. Mutually agreed values for non-functional properties are used for configuring communication channels and middleware services, and are supervised during community operation by the monitoring facilities.

Simple error handling, such as service exception handling, is usually provided and agreed in the technology and service levels. There are also more abstract errors related to enterprise computing which manifest themselves as contract breaches. Failure handling is a community specific activity grasping both kinds of the previous failure types. Failure handling mechanisms and compensation processes have to be agreed upon between all the participants of a community.

Architectural aspects contain such properties as topology of community, composition of services into business roles and coordination of services across the community. Mismatches in architectural properties of communities can be caused by faulty assumptions about other components, connections between components or topology of the community [13]. Architecture description languages such as Wright [1], Darwin [24] or Rapide [23] have been developed for defining software architectures. These languages formalise architectural properties, thus making it possible to automate validation of architectural interoperability.

Standardisation of business community architectures and business cases has also been used for providing architectural interoperability. This is the approach taken for example in ebXML [19] or RosettaNet [35].

6 Organisational Level Interoperability

Pragmatic interoperability is established at the organisational level. Properties stemming from the business level aspects such as strategies, legislation and intentions of different organisations manifest themselves at the organisational level as business rules and organisational policies. Organisational level interoperability is established by negotiation and monitoring facilities during community breeding and operation.

Business rules are declarative statements that define or constraint some aspect of a business [12]. They are part of the organisation's business knowledge which direct and influence the behaviour of an organisation [2]. Typical examples of business rules are different kinds of service pricing policies or regulations on service availability based on customer classifications. A business rule may affect the non-functional or behavioural properties of services by constraining the possible values of attributes or by introducing new kind of behaviour during service operation. To achieve automated validation of business rule interoperation, the business rules should be expressed using a feasible logical framework. For example conceptual graphs [42] and defeasible logic [2] have been used for modeling of business rules.

Organisational policies declare autonomic intentions of organisations and they are specified through the concepts of obligation, permission and prohibitions [40]. An obligation expresses that certain behaviour is required whereas permissions and prohibitions express allowable behaviour. Policies may thus modify behaviour of services by requiring certain actions to be taken instead of the others, or by prohibiting certain actions.

When organisational policies of collaborating participants are known beforehand, policy conflicts can be identified before community operation. If behavioural descriptions are given using an appropriate logic, interoperation of organisation policies with respect to the behavioural descriptions can be verified for example with model checking. However, organisational policies are inherently dynamic entities and not even necessarily published outside the organisations. Organisational policies are one of the primary causes for the dynamism in enterprise computing environments.

7 Conclusion

This paper analysed and identified different aspects of interoperability in service oriented enterprise computing environments. Different collaboration models, namely integrated, unified and federated, have been used for implementing distributed computation systems. Each of these collaboration models possess characteristic solution methods, such as common computing environments or shared

meta-information. Federated community model was identified as the most appropriate collaboration model, as this model supports the heterogeneity, autonomy and dynamism requirements inherent for this kind of environment. This support is provided by additional infrastructure services such as meta-information repositories and monitoring facilities [21], as well as negotiation mechanisms and collaboration contracts.

Interoperability was analysed using five different levels of abstraction. Division into different levels was based on the abstraction level of the concepts to be agreed upon, as well as on the subjects of interoperation. Abstraction levels were named as technology, service, community, organisation and business levels. Each of these levels contain several aspects which must be considered when establishing interoperability. For example when establishing interoperability in service level, the syntactic, semantic and behavioural properties of services must be examined. After identification of interoperability aspects, a discussion about the interoperability aspects in service, community and organisational level was given. Methods and mechanisms for establishing interoperation with respect to each aspect were briefly described.

Acknowledgement

This article is based on work performed in the web-Pilarcos project at the Department of Computer Science at the University of Helsinki. Project was funded by the National Technology Agency TEKES in Finland and Tellabs with active partners VTT, Elisa and SysOpen. The work much integrates with RM-ODP standards work, and recently has found an interesting context in INTEROP NoE collaboration.

References

1. R. Allen and D. Garlan. Formalizing architectural connection. In *ICSE '94*, pages 71–80, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.
2. G. Antoniou and M. Arief. Executable declarative business rules and their use in electronic commerce. In *SAC '02: ACM Symposium on Applied computing*, pages 6–10, New York, NY, USA, 2002. ACM Press.
3. Athena Integrated Project, EU FP6. http://www.athena-ip.org/index.php.
4. R. Bastide, P. Palanque, O. Sy, and D. Navarre. Formal specification of CORBA services: experience and lessons learned. In *Proceedings of OOPSLA '00*, pages 105–117, New York, NY, USA, 2000. ACM Press.
5. A. Bracciali, A. Brogi, and C. Canal. Dynamically Adapting the Behaviour of Software Components. In F. Arhab and C. Talcott, editors, *COORDINATION 2002*, volume 2315 of *LNCS*, pages 88–95. Springer-Verlag Heidelberg, 2002.
6. J. Brant, B. Foote, R. E. Johnson, and D. Roberts. Wrappers to the rescue. In *ECOOP'98*, volume 1445 of *LNCS*, pages 396–417. Springer-Verlag, 1998.
7. C. Canal, L. Fuentes, E. Pimentel, J. M. Troya, and A. Vallecillo. Extending CORBA Interfaces with Protocols. *The Computer Journal*, 44(5):448–462, Oct. 2001.

8. New issues in object interoperability. In A. M. J. Malenfant, S. Moisan, editor, *Object-Oriented Technology: ECOOP 2000 Workshops, Panels, and Posters*, volume 1964 of *LNCS*. Springer-Verlag GmbH, 2000.
9. J. Fang, S. Hu, and Y. Han. A service interoperability assessment model for service composition. In *IEEE International Conference on Services Computing (SCC 2004)*, pages 153–158. IEEE, 2004.
10. R. Fatoohi, V. Gunwani, Q. Wang, and C. Zheng. Performance evaluation of middleware bridging technologies. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 34–39. IEEE, 2000.
11. D. S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. OMG Press, 2003.
12. G. Fu, J. Shao, S. Embury, W. Gray, and X. Liu. A framework for business rule presentation. In *12th International Workshop on Database and Expert Systems Applications*, pages 922–926, 2001.
13. D. Garlan, R. Allen, and J. Ockerbloom. Architectural mismatch or why it's hard to build systems out of existing parts. In *ICSE '95*, pages 179–185, New York, NY, USA, 1995. ACM Press.
14. IEC. *TC65/290/DC: Device Profile Guideline*, Mar. 2002.
15. INTEROP NoE, EU FP6. http://tmitwww.tm.tue.nl/research/Interop-NoE.html.
16. ISO. *ISO 14258 – Concepts and rules for enterprise models*. ISO TC184 SC5 WG1, Apr. 1999.
17. S. Jha, J. Palsberg, and T. Zhao. Efficient Type Matching. *Lecture Notes in Computer Science*, 2303:187–206, 2002.
18. D. Konstantas. Object oriented interoperability. In *ECOOP '93 - Object-Oriented Programming: 7th European Conference*, volume 707 of *LNCS*, pages 80–102. Springer-Verlag GmbH, 1993.
19. A. Kotok and D. R. R. Webber. *ebXML: The New Global Standard for Doing Business Over the Internet*. New Riders, Boston, 2001.
20. L. Kutvonen, J. Metso, and T. Ruokolainen. Inter-enterprise collaboration management in dynamic business networks. In *CoopIS 2005 conference*, Agya Napa, Cyprus, Oct. 2005. To be published.
21. L. Kutvonen, T. Ruokolainen, J. Metso, and J. Haataja. Interoperability middleware for federated enterprise applications in web-Pilarcos. In *INTEROP-ESA'05*, 2005.
22. D. A. Lamb. IDL: sharing intermediate representations. *ACM Trans. Program. Lang. Syst.*, 9(3):297–318, 1987.
23. D. Luckham and J. Vera. An event-based architecture definition language. *IEEE Transactions on Software Engineering*, 21(9):717–734, sep 1995.
24. J. Magee, N. Dulay, and J. Kramer. Structuring parallel and distributed programs. *Software Engineering*, 8(2):73–82, Mar. 1993.
25. S. McIlraith, T. Son, and H. Zeng. Semantic Web services. *Intelligent Systems*, 16(2):46–53, 2001.
26. M. Mecella and B. Pernici. Designing wrapper components for e-services in integrating heterogeneous systems. *The VLDB Journal*, 10(1):2–15, 2001.
27. Object Management Group. *CORBA 3.0 - OMG IDL Syntax and Semantics chapter*, jul 2002.
28. R. Orfali, D. Harkey, and J. Edwards. *Instant CORBA*. John Wiley & Sons, Inc., 1997.
29. J. Palsberg and T. Zhao. Efficient and flexible matching of recursive types. In *Logic in Computer Science*, pages 388–398, 2000.

30. M. P. Papazoglou and D. Georgakopoulos. Introduction. *Commun. ACM*, 46(10):24–28, 2003. Special issue on Service-Oriented Computing.
31. J. Peer. Bringing together semantic web and web services. In *The Semantic Web - ISWC 2002: First International Semantic Web Conference*, volume 2342 of *Lecture Notes in Computer Science*, pages 279–291. Springer-Verlag Heidelberg, 2002.
32. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.
33. E. Rahm, H.-H. Do, and S. Massmann. Matching large XML schemas. *SIGMOD Rec.*, 33(4):26–31, 2004.
34. D. Rine, N. Nada, and K. Jaber. Using adapters to reduce interaction complexity in reusable component-based software development. In *SSR '99: Symposium on Software Reusability*, pages 37–43, New York, NY, USA, 1999. ACM Press.
35. RosettaNet Consortium. Rosettanet implementation framework: Core specification v02.00.00, 2004. http://www.rosettanet.org/.
36. P. Siddhartha and S. Sengupta. Web services interoperability: A practitioner's experience. In *CoopIS/DOA/ODBASE*, pages 587–601, 2002.
37. J. Siméon and P. Wadler. The essence of XML. In *POPL '03: Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 1–13, New York, NY, USA, 2003. ACM Press.
38. M. P. Singh, A. K. Chopra, N. Desai, and A. U. Mallya. Protocols for processes: programming in the large for open systems. *SIGPLAN Not.*, 39(12):73–83, 2004.
39. M. P. Singh and M. N. Huhns. *Service-Oriented Computing: Semantic, Processes, Agents*. John Wiley & Sons, Ltd., 2005.
40. M. Steen and J. Derrick. Formalising ODP enterprise policies. In *EDOC'99*, pages 84–93. IEEE, 1999.
41. Sun Microsystems. *Java 2 Platform, Enterprise Edition (J2EE), 1.4 Specification*, 2002.
42. I. Valatkaite and O. Vasilecas. A conceptual graphs approach for business rules modeling. In *Advances in Databases and Information Systems*, volume 2798 of *LNCS*, pages 178–189, Sept. 2003.
43. A. Vallecillo, J. Hernandez, and J. M. Troya. Component Interoperability. Technical Report ITI-2000-37, University of Malaga, July 2000.
44. A. Vallecillo, J. Hernández, and J. M. Troya. Object Interoperability. In *Object-Oriented Technology. ECOOP'99 Workshop*, volume 1743 of *LNCS*, pages 1–21. Springer-Verlag Heidelberg, 1999.
45. P. Wegner. Interoperability. *ACM Computing Surveys (CSUR)*, 28(1):285–287, 1996.
46. D. M. Yellin and R. E. Strom. Protocol specifications and component adaptors. *ACM Trans. Program. Lang. Syst.*, 19(2):292–333, 1997.
47. A. M. Zaremski and J. M. Wing. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 6(4):333–369, 1997.