LANGUAGE-BASED SUPPORT FOR SERVICE ORIENTED ARCHITECTURES: FUTURE DIRECTIONS*

Pablo Giambiagi
Swedish Institute of Computer Science – P.O. Box 1263, SE-16429 Kista, Sweden

Olaf Owe, Gerardo Schneider
Dept. of Informatics, Univ. of Oslo – P.O. Box 1080 Blindern, N-0316 Oslo, Norway

Anders P. Ravn
Dept. of Computer Science, Aalborg University – Fredrik Bajers vej 7E, DK-9220 Aalborg, Denmark

Keywords: SOA, Web Services, Web Languages, Contracts.

Abstract: The popularity of service-oriented architectures (SOA) lives on the promise of dynamic IT-supported inter-business collaborations. Yet the programming models in use today are a poor match for the distributed, loosely-coupled, document-based SOA; and the gap widens: interoperability across organizations needs contracts to reduce risks. Thus, high-level contract models are making their way into SOA, but application developers are still left to their own devices when it comes to writing code that will comply with a contract. This paper surveys existing and future directions regarding language-based solutions to the above problem.

1 INTRODUCTION

In the past, advocates of the service-oriented architecture (SOA) have predicted that the successful integration of loosely-coupled services belonging to different, sometimes competing, but always collaborating organizations, would storm the world. It would create a myriad of new business opportunities, enabling the formation of virtual organizations where SMEs would join forces to thrive in ever increasingly competitive global markets. Yet the industry has been slow to deploy its SOAs, and the degree of integration between different organizations remains low.

At the moment the developer faces a situation where the tools originally used to produce intra-organizational, non-distributed applications are already overstretched to cope with issues of distribution across organizational domains. Furthermore, collaboration presumes a minimum level of mutual trust, and wherever trust is not considered sufficient, businesspeople turn to contracts as a mechanism to reduce risks. In other terms, for the SOA to deliver its promised advantages, developers need not only language support for distribution, but also cost effective contract management solutions. Researchers and industries alike have began addressing this very essential issue with a top-down approach. Several electronic contract languages, their models and reasoning techniques are in the process of being discussed and refined. Thus we see a pressing need to provide the actual system developers with the means to implement services that meet the requirements dictated by such contracts.

In the next section, we recall the main features of SOAs and discuss the requirements posed by contracts. In Sec. 3, we discuss programming languages and SOA. In Sec. 4, we identify open problems and in Sec. 5 we discuss possible concrete scenarios for addressing them.

2 SOA AND CONTRACTS

In a SOA, applications are essentially distributed systems composed of services (Fig. 1, borrowed from (Papazoglou, 2003)). A service is a loosely-coupled, technology neutral and self-describing computation element. Loose coupling is achieved through encapsulation and communication through message passing; technology neutrality results from adopting standardized mechanisms; and rich interface languages permit the service to export sufficient information so that eventual clients can discover and connect to it (Papazoglou, 2003). A SOA can be implemented in many different ways, e.g. using *web services*.

*Supported by the Nordunet3 Project "Contract-Oriented Software Development for Internet Services".

Giambiagi P., Owe O., Schneider G. and P. Ravn A. (2006).
LANGUAGE-BASED SUPPORT FOR SERVICE ORIENTED ARCHITECTURES: FUTURE DIRECTIONS.
In *Proceedings of the First International Conference on Software and Data Technologies*, pages 339-344
DOI: 10.5220/0001315103390344

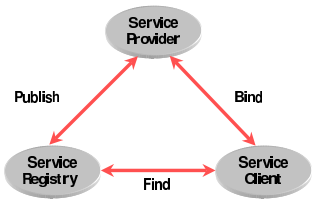

Figure 1: The basic Service Oriented Architecture.

Web services exchange SOAP messages over standard Internet protocols which carry a payload built from a stack of open XML standards (WSA, 2004). There are strong similarities between services and components in a component-based system (Szyperski, 2003). However, services usually have a coarser granularity and the communication medium with its high latency and openness constrains reliability and security in ways that easily go beyond what can be found in most component-based systems.

The services in a SOA usually belong to different organizational domains and therefore there is no single line of authority regulating their interactions. In principle a consumer must trust the provider to deliver the expected service, or establish a contract with it. For our purposes, a contract describes an agreement between distinct services that determines rights and obligations on its signatories, and for which there exists static or dynamic ways of identifying contract violations.

In the case of a bilateral contract, one usually talks about the roles of *service provider* and *service consumer*; but multi-lateral contracts are also possible where the participants may play other roles. A service provider may also use a contract template (i.e. a yet-to-be-negotiated contract) to publish the services it is willing to provide. As a service specification, a contract may describe many different *aspects* of a service, including functional properties and also non-functional properties like security, quality of service (QoS) and reputation.

Contract Models There exists a number of contract models for services. The business process standard ebXML[2] describes a Collaboration Protocol Agreement as a contract between business partners that specifies the behavior of each service (by simply stating its role) and how information exchanges are to be encoded. IBM's Web Service Level Agreement[3] is an XML specification of performance constraints associated with the provision of a web service. It defines the sources of monitoring data, a set of metrics (i.e. functions) to be evaluated on the data, and obligations on the signatories to maintain the metric values within certain ranges. The set of predefined metrics and the structure of WSLA contracts are designed for services involving job submissions in a grid computing environment. The later WS-Agreement[4], a Global Grid Forum recommendation that has not reached the standard status yet, is based on WSLA, but adapted to more recent web-services standards, e.g. WS-Addressing and WS-Resource Framework. WS-Agreement is also parametric on the language used to specify the metrics.

A number of problems have previously been identified for these standards and specifications: They are restricted to bilateral contracts, lack formal semantics (and therefore it is difficult to reason about them), their treatment of functional behavior is rather limited and the sub-languages used to specify QoS and security constraints are usually limited to small application-specific domains. In order to remedy the situation researchers have produced contract taxonomies (Aagedal, 2001; Beugnard et al., 1999; Tosic, 2005), formalizations using logics, e.g. classical (Davulcu et al., 2004), modal (Daskalopulu and Maibaum, 2001), deontic (Paschke et al., 2005b) and defeasible logic (Governatori, 2005), and formalizations based on models of computation –e.g. finite state machines (Molina-Jimenez et al, 2004) and Petri nets (Daskalopulu, 2000). The diversity of contract types, their applications and properties poses a serious challenge to the definition of a *generic contract model*. This, however, has been identified as a major precondition for the advancement of the area (Bouyssounouse and Sifakis, 2005).

Discovery and Negotiation In a setup for contract-enhanced service provision, providers are expected to make service descriptions available for consumers to discover and choose among them. The description takes the form of a proto-contract, or template, setting the basis for negotiating the provision of the service. Specifications like ebXML and WS-Agreement define sub-languages for such contract templates, though they are usually attached to a very specific negotiation model. There is, however, a large body of research on contract negotiation protocols under different threat models, particularly in the area of agent-based systems (Andreoli and Castellani, 2001; Picard, 2003; Kraus, 2001).

Monitoring Monitoring presents an important list of challenges. First, monitoring data (including execution events and samplings of continuous processes) needs to be collected in a timely, reliable and trustworthy manner even within a distributed system. Moreover, monitors are usually weaved into the application code by specialists (not by ordinary program-

[2] `http://www.ebxml.org`.

[3] WSLA, `http://www.research.ibm.com/wsla/`.

[4] `https://forge.gridforum.org/projects/graap-wg/`.

mers), creating complex dependencies that seriously affect the software development process.

Quality of Service According to the ARTIST roadmap (Bouyssounouse and Sifakis, 2005), quality of service is a "function mapping a given system instance with its full behavior onto some [quantitative] scale". Typical QoS measures for web services include *average response time*, *minimum communication bandwidth* and *peak CPU usage*. QoS measures usually depend on the behavior of the environment as well as of the service, thus models tend to have a stochastic nature, although this is not really necessary for monitoring purposes.

Typically, contract languages for QoS of web services consist of three main sub-languages. Their purpose is to specify: (1) The QoS measures (i.e. functions) including their domains; (2) a mapping between elements in the execution model (e.g. observable events) and the domains of QoS measures; and (3) the constraints on QoS measurements (i.e. the obligations). The design of such languages is therefore centered around the concept of QoS measure function. However, realistic contracts are not easily modeled as a set of functions: they are built upon the fundamental concept of obligation, to which other concepts (like QoS measures) become accessory. For instance, the fulfillment or violation of an obligation may trigger other obligations. Function-based approaches need then to encode *obligation performances* as *QoS measures*. Moreover, the inclusion of time considerations becomes unnecessarily complicated.

Security So far, contract languages for security pay almost exclusive attention to access control issues – e.g. (de Win et al, 2005)– but it is evident that they should be enhanced to cover other areas of security such as integrity and confidentiality.

3 SOA AND LANGUAGES

Current programming language abstractions are not adequate for SOA, much less for web-service development. The industry develops web services using the object-oriented programming (OOP) paradigm which maps badly to document-based communication, i.e. SOAP-transported XML documents, required by web-services (Meijer et al., 2003). Besides, many current production OOP languages (e.g. Java and C#) are based on the shared-state model so they do not handle concurrency and message passing particularly well. Another criticism of OOP concerns *reusability*. Object-orientation provides two distinct mechanisms for composing concerns, namely aggregation and inheritance, which may be difficult to combine with synchronization, history information or multiple views. Thus OOP needs better abstraction mechanisms.

The programming language community has long identified the need to provide easier ways to extend the abstraction mechanisms of a language. One of the main approaches is that of aspect-oriented programming (AOP) (Filman et al., 2005), which helps separate cross-cutting concerns (like access control) from the main business logic. AOP is composed of a set of techniques, including code instrumentation and runtime interceptors.

A similar approach uses composition filters (CF) (Aksit et al., 1992), where the idea is not to replace the programming paradigm but to enhance the expressive power and maintainability of current object-oriented (OO) languages. CF may be considered a modular extension to the OO model with *interface* layers including the so-called *filters*. Advantages of CFs with respect to aspects are exposed in (Bergmans and Aksit, 2001).

An alternative approach defines new kinds of languages that adapt themselves better to the challenges posed by web services. Some concentrate on bridging the gap between the program language and the XML objects that web services should exchange (Florescu et al., 2002), others provide abstractions to manipulate interfaces (Cooney et al., 2005), and others address asynchronous communication by means of message passing –e.g., C_ω (Bierman et al., 2005). The latter combines features from two other research languages: (a) Polyphonic C# (Benton et al., 2004): a control flow extension with asynchronous wide-area concurrency, and (b) Xen (Meijer et al., 2003): a data type extension for processing XML and table manipulation. Along the same lines, the Xtatic project (Gapeyev and Pierce, 2003) aims at extending C# with native XML processing adding *regular types* and *regular patterns*. More oriented to web service development, a new language is proposed in (Cooney et al., 2005) which combines XQuery's semantics with imperative constructs and a join calculus-style concurrency model. This proposal solves some of the problems of mainstream languages (e.g., Java and C#) like concurrency and message correlation problems. It lacks, however, useful features like interface inheritance, correlated messages and garbage collection. Furthermore, the current implementation assumes a shared-state concurrency model.

Another interesting language is Creol (Johnsen and Owe, 2004), whose programs consist of concurrent objects with internal process control communicating asynchronously. By means of mechanisms for conditional processor release points, passive waiting, and time-out, explicit synchronization primitives are not needed in the language. Compared to for instance Polyphonic C#, Creol has a simpler set of communi-

cation primitives, using the concept of asynchronous method call, while maintaining multiple inheritance. It also offers a synchronized merge operator which effectively reduces the problems related to the so-called inheritance anomaly (Matsuoka and Yonezawa, 1993), while allowing compositional reasoning.

The ideal language for SOA and web-services development seems to be one which combines the advantages of the above-mentioned languages. It should be based on asynchronous communications with facilities for providing good abstractions for manipulating interfaces, XML processing and web service development. The inclusion of regular types *a la* XDuce (Hosoya and Pierce, 2003) and CDuce (Benzaken et al., 2003) would be a plus.

Programming and Contracts The solutions mentioned so far still lack support for discovery, monitoring and management of contracts. Approaches like AOP and CF can potentially provide some help here (Becker and Geihs, 2001), but they fail to abstract low-level issues and basically leave too much freedom to the programmer (which leads to code maintenance and analysis issues). This is an issue for which no consolidated work has been completed yet. Instead, one should look for different bits and pieces, depending on the characteristics of the contract.

Although contracts could cover many other aspects, guarantees of the timely provision of services are commonplace in SOA. Despite of the current wide acceptance of AOP as a good paradigm for improving reusability and modularity, there is no convincing solution to the application of aspects to real-time systems. In some cases (Tsang et al., 2004), aspect-orientation seems to perform better than object-orientation when dealing with real-time specifications, regarding system properties such as testability and maintainability. On the other hand, in (Assayad et al., 2005), there is a formal framework for multi-threaded and multi-processor software synthesis using timing constraints, where it is shown that AOP is not suitable for such cases.

A new concept for real-time system development (ACCORD) is presented in (Tesanovic et al., 2004), combining component-based and aspect-oriented software development. ACCORD bridges the gap between modern software engineering methods –focused mainly on component models, interfaces and separation of concerns– and real-time design methods. The focus is primarily on a design methodology, but not on *analysis* and verification of real-time systems. It is not clear, either, how the methodology could be used to develop asynchronous open distributed systems.

Programs using real-time features are, in general, difficult to design and verify, even more when combined with an inheritance mechanism. Changing application requirements or real-time specifications in real-time OO languages may produce unnecessary redefinitions. This is called the *real-time specification inheritance anomaly*. To solve it, (Aksit et al., 1994) propose real-time composition filters. Similar ideas could be applied to support contract-based system design.

A contribution towards verifying properties of contracts involving real-time as formulated in existing languages is found in (Diaz et al., 2005). They use a translation to a real-time model checker to verify the cooperation aspect of contracts.

In conclusion, there is still plenty of work to do in directly supporting development of services that can be trusted to implement their contracts.

4 RESEARCH DIRECTIONS

The main problems and open issues identified for supporting web services development include:

- Formal definition of generic contracts, in particular for QoS and security.
- Negotiable and monitorable contracts. Contracts must be negotiated till both parts agree on their final form, and they must be monitorable in the sense that there must be a way to detect violations. No existing programming language supports negotiable and monitorable contracts.
- Combination of object-orientation and concurrency models based on asynchronous message-passing. The shared-state based concurrency model is not suitable for web service development.
- Integration of XML into a host language, reducing the distance between XML and object data-models.
- Harmonious coexistence at the language level of real-time and inheritance mechanisms.
- Verification of contract properties. The integration of contracts in a programming language should be accompanied by good support for guaranteeing essential properties. Guaranteeing the non-violation of contracts might be done in (at least) four different ways: 1. with runtime enforcement, e.g. through monitors; 2. by construction, e.g. through low-level language mechanisms; 3. with standard static program analysis techniques; or 4. through model checking. None of the above can be used as a universal tool; they must be combined.

Addressing these issues and problems, we need to develop a model of contracts in a SOA that is broad enough to cater for at least contracts for QoS and security. A minimum requirement is the ability to seamlessly combine real-time models (for QoS specifications) and behavioral models (essential to constrain protocol implementation and to enforce security). Contract models should also address discovery

and negotiation. Yet, the formal definition of contracts should be only a first step towards a more ambitious task, namely to provide language-based support for programming and effective usage of such contracts. Some contracts may be seen as a *wrapper* which "envelopes" the code/object under the scope of the contract. *Firewalls*, for instance, may be seen as a kind of contract between the machine and the external applications wanting to run on that machine. It would be interesting to investigate a language primitive to create wrapped objects which are correct-by-construction. On the other hand, contracts for QoS and security could be modeled as first-class entities using a "behavioral" approach, through interfaces. In order to tackle timed constraints (related to QoS) such interfaces need also to incorporate time. As exposed in the ARTIST road-map, finding languages or notations for describing timing behaviors and timing requirements is easy; the real challenges are in analysis. Besides syntactic extensions, the language needs to have timing semantic extensions in order to allow extraction of a timed model, e.g. a timed automaton. This model may be checked with existing tools, e.g. Kronos (Yovine, 1997) and Uppaal (Bengtsson et al., 1995). Model checking tools will help to prove real-time properties, like guaranteeing that a service will satisfy its promised response-time constraints. Other properties may, instead, be proved to be correct-by-construction (e.g. wrappers).

In practice, many properties can only be proved correct through runtime approaches. A promising direction is to develop techniques for constructing runtime monitors from contracts, which will be used to enforce its non-violation.

5 FINAL DISCUSSION

The web is mostly used nowadays for retrieving remote information, but there is a high demand for more challenging applications that offer, negotiate and discover web services through XML interfaces. This new direction requires redesigning software architectures and revising the existing foundations of computer science (Montanari, 2004). Moreover, in order to make collaboration a reality among different web-services, the formal definition of monitorable and negotiable contracts has become imperative. In this paper we have surveyed the main current approaches to programming in a SOA and their support in state-of-the-art programming languages. We have identified some problems and open issues of current approaches and we have proposed general research directions.

We believe object-orientation could still be a good paradigm for modeling open distributed systems, since its main problems come from sequential language design and implementation decisions, not from its original philosophy.

Regarding the formal definition of contracts, we believe such a generic model can be described harmoniously using real-time extensions of rewriting logic (Ölveczky and Meseguer, 2002). This is in line with recent investigations in the use of rule languages to model contracts (Grosof and Poon, 2002; Paschke et al., 2005a). While these languages are essentially ad-hoc, we expect to profit from the existing large body of research in rewriting logics. The rule-based approach brings along new challenges in the definition of appropriate negotiation schemes (Reeves et al., 2001; Paschke et al., 2005c). Here again, rewriting logic can give invaluable help. Its reflection and meta-level computation properties may help define and structure the negotiation protocol.

Concerning the choice of a host language for incorporating contracts as first citizens, we believe Creol might be a good starting point. The Creol project has addressed many of the problems of OOP and it has a formal semantics defined in rewriting logic. More details on such a line of research can be found in the technical report (Giambiagi et al., 2006).

REFERENCES

Aagedal, J. (2001). *Quality of Service Support in Development of Distributed Systems*. PhD thesis, Dept. of Informatics, University of Oslo.

Aksit, M., Bergmans, L., and Vural, S. (1992). An object-oriented language-database integration model: The composition-filters approach. In *ECOOP*, pp. 372–395.

Aksit, M., Bosch, J., van der Sterren, W., and Bergmans, L. (1994). Real-time specification inheritance anomalies and real-time filters. In *ECOOP*, vol. 821 of *LNCS*, pp. 386–407.

Andreoli, J. M. and Castellani, S. (2001). Towards a Flexible Middleware Negotiation Facility for Distributed Components. In *DEXA*, pp. 732–736. IEEE C.S.

Assayad, I., Bertin, V., Defaut, F.-X., Gerner, P., Quevreux, O., and Yovine, S. (2005). Jahuel: A formal framework for software synthesis. In *ICFEM*, vol. 3785 of *LNCS*, pp. 204–218.

Becker, C. and Geihs, K. (2001). Quality of Service and Object-Oriented Middleware-Multiple Concerns and their Separation. In *ICDCSW*.

Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., and Yi, W. (1995). UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In *HS III*, vol. 1066 of *LNCS*, pp. 232–243.

Benton, N., Cardelli, L., and Fournet, C. (2004). Modern concurrency abstractions for c#. *ACM Trans. Program. Lang. Syst.*, 26(5):769–804.

Benzaken, V., Castagna, G., and Frisch, A. (2003). Cduce: an xml-centric general-purpose language. *SIGPLAN Not.*, 38(9):51–63.

Bergmans, L. and Aksit, M. (2001). Composing cross-cutting concerns using composition filters. *Commun. ACM*, 44(10):51–57.

Beugnard, A., Jzquel, J.-M., and Plouzeau, N. (1999). Making components contract aware. *IEEE*, 32(7):38–45.

Bierman, G., Meijer, E., and Schulte, W. (2005). The essence of data access in Cω. In *ECOOP*, vol. 3586 of *LNCS*, pp. 287–311.

Bouyssounouse, B. and Sifakis, J., editors (2005). *Embedded System Design: The ARTIST Roadmap for Research and Development*, vol. 3436 of *LNCS*.

Cooney, D., Dumas, M., and Roe, P. (2005). A programming language for web service development. In *ACSC*, vol. 38 of *CRPIT*, pp. 143–150.

Daskalopulu, A. (2000). Model checking contractual protocols. In *Legal Knowledge and Information Systems, Jurix 2000*, IOS Press, pp. 35–47.

Daskalopulu, A. and Maibaum, T. S. E. (2001). Towards Electronic Contract Performance. In *DEXA*, pp. 771–777. IEEE.

Davulcu, H., Kifer, M., and Ramakrishnan, I. V. (2004). CTR-S: A Logic for Specifying Contracts in Semantic Web Services. In *WWW*, pp. 144–153.

de Win, B., Piessens, F., Smans, J. and Joosen, W. (2005). Towards a unifying view on security contracts. In *SESS*, pp. 1–7. ACM Press.

Diaz, G., Pardo, J.-J., Cambronero, M. E., Valero, V., and Cuartero, F. (2005). Verification of web services with timed automata. In *WWV*, pp. 177–191.

Filman, R. E., Elrad, T., Clarke, S., and Akşit, M., editors (2005). *Aspect-Oriented Software Development*. Addison-Wesley, Boston.

Florescu, D., Grünhagen, A., and Kossman, D. (2002). XL: An XML programming language for web service specification and composition. In *WWW*, pp. 65–76.

Gapeyev, V. and Pierce, B. (2003). Regular object types. In *ECOOP*, vol. 2743 of *LNCS*, pp. 151–175.

Giambiagi, P., Owe, O., Ravn, A. P., and Schneider, G. (2006). Contract-based internet service software development: A proposal. Technical Report 333, Dept. of Informatics, University of Oslo, Norway.

Governatori, G. (2005). Representing business contracts in RuleML. *Int. Journal of Coop. Inf. Sys.*, 14:181–216.

Grosof, B. and Poon, T. (2002). Representing Agent Contracts with Exceptions using XML Rules, Ontologies, and Process Descriptions. In *RuleML*.

Hosoya, H. and Pierce, B. C. (2003). Xduce: A statically typed xml processing language. *ACM Trans. Inter. Tech.*, 3(2):117–148.

Johnsen, E. B. and Owe, O. (2004). An asynchronous communication model for distributed concurrent objects. In *SEFM*, pp. 188–197. IEEE.

Kraus, S. (2001). Automated Negotiation and Decision Making in Multiagent Environments. 2086:150–172.

Matsuoka, S. and Yonezawa, A. (1993). Analysis of inheritance anomaly in object-oriented concurrent programming languages. *Research directions in concurrent object-oriented programming*, pp. 107–150.

Meijer, E., Schulte, W., and Bierman, G. (2003). Programming with cirdvcles, triangles and rectangles. In *XML Conference*.

Molina-Jimenez, C., Shrivastava, E. S. and Warne, J. (2004). Run-time Monitoring and Enforcement of Electronic Contracts. *Elect. Commerce Research and Applications*, 3(2).

Montanari, U. (2004). Web services and models of computation. In *WS-FM*, vol. 105 of *ENTCS*, pp. 5–9.

Papazoglou, M. P. (2003). Service-Oriented Computing: Concepts, Characteristics and Directions. In *WISE*, pp. 3–12. IEEE.

Paschke, A., Bichler, M., and Dietrich, J. (2005a). ContractLog: An Approach to Rule Based Monitoring and Execution of Service Level Agreements. In *RuleML*, vol. 3791 of *LNCS*, pp. 209–217.

Paschke, A., Dietrich, J., and Kuhla, K. (2005b). A Logic Based SLA Management Framework. In Semantic Web and Policy Workshop 2005, pp. 68–83.

Paschke, A., Kiss, C., and Al-Hunaty, S. (2005c). A Pattern Language for Decentralized Coordination and Negotiation Protocols. In *EEE*, pp. 404–407. IEEE.

Picard, W. (2003). NeSSy: Enabling Mass E-Negotiations of Complex Contracts. In *DEXA*, pp. 829–833. IEEE.

Reeves, D. M., Wellman, M. P., and Grosof, B. N. (2001). Automated negotiation from declarative contract descriptions. In *AGENTS*, pp. 51–58. ACM Press.

Szyperski, C. (2003). Component technology - what, where, and how? In *ICSE*, pp. 684–693. IEEE.

Tesanovic, A., Nystrm, D., Hansson, J., and Norstrm, C. (2004). Aspects and components in real-time system development: Towards reconfigurable and reusable software. *Journal of Embedded Computing*, 1(1).

Tosic, V. (2005). On Comprehensive Contractual Descriptions of Web Services. In *EEE*, pp. 444–449. IEEE.

Tsang, S. L., Clarke, S., and Baniassad, E. L. A. (2004). An evaluation of aspect-oriented programming for java-based real-time systems development. In *ISORC*, pp. 291–300.

WSA (2004). Web Services Architecture. W3C Working Group Note, `www.w3.org/TR/ws-arch/`.

Yovine, S. (1997). Kronos: A verification tool for real-time systems. *Int. Journal of Software Tools for Tech. Transfer*, 1(1/2):123–133.

Ölveczky, P. and Meseguer, J. (2002). Specification of real-time and hybrid systems in rewriting logic. *TCS*, 285(2):359–405.