UNIVERSITY OF OSLO
Department of Informatics

Contract-based Internet Service Software Development: A Proposal

Research Report No. 333

Pablo Giambiagi

Olaf Owe

Gerardo Schneider

Anders P. Ravn

Isbn 82-7368-288-9
Issn 0806-3036

January 2006

Contract-based Internet Service Software Development: A Proposal

Pablo Giambiagi[*] Olaf Owe[†] Gerardo Schneider[‡]
Anders P. Ravn[§]

January 2006

Abstract

The fast evolution of the Internet has popularized service-oriented architectures dynamic IT-supported inter-business collaborations. Yet, interoperability between different organizations, requires contracts to reduce risks. Thus, high-level models of contracts are making their way into service-oriented architectures, 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 proposes new language-based solutions to the above problem. Contracts are formalized as behavioral interfaces, and abstraction mechanisms may guide the developer in the production of contract-aware applications. We concentrate on contracts dealing with performance (real-time) and information flow (confidentiality).

1 Introduction

Already several years ago, technology gurus predicted that the next big trend in software system development would be the service-oriented architecture, SOA. A successful integration of loosely-coupled services belonging to different, sometimes competing, but always collaborating organizations would

[*]SICS, P.O. Box 1263, SE-16429 Kista, Sweden. E-mail: pablo@sics.se

[†]Dept. of Informatics – Univ. of Oslo, P.O. Box 1080 Blindern, N-0316 Oslo, Norway. E-mail: olaf@ifi.uio.no

[‡]Dept. of Informatics – Univ. of Oslo, P.O. Box 1080 Blindern, N-0316 Oslo, Norway. E-mail: gerardo@ifi.uio.no

[§]Dept. of Computer Science – Aalborg University, Fredrik Bajers vej 7E, DK-9220 Aalborg, Denmark. E-mail: apr@cs.aau.dk

storm the world. It would create a myriad of new business opportunities, enabling the formation of virtual organizations where SMEs[1] would join forces to thrive in ever increasingly competitive global markets. While the dream lives on, and the industry develops and deploys web services, the degree of integration achieved between different organizations remains low. 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 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. While this is a natural approach, we see the absolute need to provide the actual system developer with the means to implement their services to meet the requirements dictated by the contracts.

At the moment the developer faces a situation where the programming languages originally used to produce intra-organization, non-distributed applications are already overstretched to cope with issues of distribution across organizational domains. When it comes to contracts, the abstraction mechanisms of current languages give almost no assistance to the developer. Therefore we propose to use a richer language, based on the concepts of Creol [18], which allows formal verification of requirements of a contract to be done or even automated using the Maude tool [38].

1.1 Related Work

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 of the day is that of Aspect-Oriented Programming (AOP) [26], which helps separate cross-cutting concerns (like logging and 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) [2], where the idea is not to replace the programming paradigm but to enhance the expressive power and maintainability of current object-oriented languages. CF may be considered as a modular extension to the object-oriented model with *interface* layers including the so-called *filters*. Advantages of CFs with respect to aspects are exposed in [12].

An alternative approach aims at defining new kinds of languages that

[1]SME: small and medium enterprise.

adapt themselves better to the challenges posed by web services. Some concentrate on bridging the gap between the program language objects and the XML objects that web services should exchange [27, 28, 39], others provide abstractions to manipulate interfaces [17], and others address asynchronous communication by means of message passing [14]. In [17], for instance, a new language proposal has been presented, which combines XQuery's semantics with imperative constructs and a join calculus-style concurrency model. The proposed language seems to solve some of the problems of main stream languages, like concurrency and message correlation problems, which arises for instance in Java and C#. It lacks, however, useful features likeinterface inheritance and the current implementation is based on the shared-state concurrency and does not includes correlated messages nor garbage collection.

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 (see e.g. [10]), 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).

Despite of the current wide acceptance of AOP as a good paradigm for improving reusability and modularity, there is no convincing and final solution to the application of aspects to real-time systems. In some cases [55], aspect-orientation seems to perform better than object-orientation when dealing with real-time specification, regarding system properties such as testability and maintainability. On the other hand, in [7], there is a formal framework for multi-threaded software and multi-processor architecture software synthesis using timing constraints, where it is shown that aspect-oriented software development is not suitable for such cases.

A new concept for real-time system development (ACCORD) is presented in [53], combining both component-based and aspect-oriented software development (CBSD and AOSD, respectively). ACCORD bridges the gap between modern software engineering methods –focused mainly on component models, interfaces and separation of concerns– and real-time design methods, by proposing a model for software development using the advantages of both communities. As far as we know, the focus is primarily on the design methodology of real-time systems by using CBSD and AOSD, but not on *analysis* (e.g. verification) of real-time systems. It is not clear, either, how the methodology could be used in asynchronous open distributed systems such as the Internet.

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 object-oriented languages may produce unnecessary redefinitions. This is called the *real-time specification inheritance anomaly.* To our knowledge, [3] is the only work trying to solve this problem; it does so by proposing real-time composition filters. The idea seems attractive and could be incorporated within a contract-based approach.

A contribution towards verifying properties of contracts involving real-time as formulated in existing languages is found in [24, 23]. 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.

1.2 Overview

In the following section, we introduce Service Oriented Architectures (SOA) and Contracts. In Section 3, we discuss Programming Languages and SOA implementation. In Section 4, we identify open problems. In Section 5 we outline our research agenda while Section 6 concludes on its feasibility.

2 Service-Oriented Architectures

In a Service-Oriented Architecture (SOA), applications are essentially distributed systems composed of services (see Fig. 1, borrowed from [44]). 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 [44].

A SOA can be implemented in many different ways. A currently very popular approach uses a specific kind of service called *web service*. *Web services* exchange SOAP [51] messages over standard Internet protocols (e.g. HTTP) which carry a payload built from a stack of open XML standards [58]. There are strong similarities between services and components in a component-based system [52]. However, services usually have a coarser granularity and the communication medium (the Internet) 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.

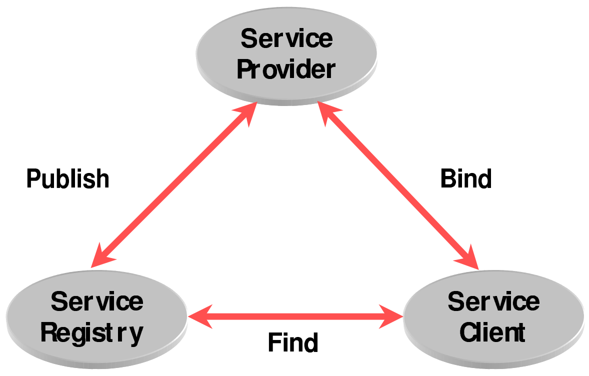

Figure 1: The basic Service Oriented Architecture

2.1 Contracts

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 purpose, a contract is a generic term for the specification of a service which is negotiable and either statically enforceable or monitorable. In other words, a contract describes an agreement between distinct services that determines rights and obligations on its signatories, and for which there exists a programmatic way 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 (i.e. behavior) and also non-functional properties like security (e.g. access control), quality of service (QoS), information flow and reputation.

Following [13], contracts may be classified in four levels[2]:

"The first level, basic, or syntactic, contracts, is required sim-

[2]This classification refers to level 2 contracts as "behavioral contracts". When we use the same name in the rest of the document we actually mean level 4 contracts. The reader should be aware that from now on, when we refer to "behavioral contracts" we are not restricted to sequential systems and mean level 4 contracts.

ply to make the system work. The second level, behavioral contracts, improves the level of confidence in a sequential context. The third level, synchronization contracts, improves confidence in distributed or concurrency contexts. The fourth level, quality-of-service contracts, quantifies quality of service and is usually negotiable."

2.1.1 Contract Models

There exists a number of contract models for services. The business process standard ebXML [25] 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 (WSLA [60]) 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 [59], 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; but it must be an XML dialect.

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 the research community has produced contract taxonomies [1, 13, 54], formalizations using logics (e.g. classical [22], modal [21], deontic [46] and defeasible logic [31]) and formalization based on models of computation (e.g. finite state machines [16] and Petri Nets [20]). 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 [15].

2.1.2 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 [6, 48, 35].

2.1.3 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. A set of collaborating Internet services forms a distributed system, and so must be the monitoring subsystem itself, with the consequent difficulties regarding coordination and dependability. Moreover, monitors are usually weaved into the application code by specialists (not by ordinary programmers), creating complex dependencies that seriously affect the software development process.

2.1.4 Quality of Service

According to the ARTIST road-map [15], 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*. Contract languages like WSLA and WS-Agreement permit specification of QoS constraints for web services. 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 Internet 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 these contract languages is therefore centered around the concept of QoS measure. However, realistic contracts are not easily modeled as a set of functions. Instead, 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 elements in the domains of QoS measures.

The inclusion of time scales into these domains also complicates the design in ways we consider unnecessary. For example, WSLA and WS-Agreement use the concept of *time series* to define time points where measurements need to be collected and then aggregated.

2.1.5 Information Flow

Information flow concerns issues like confidentiality and integrity of information. Contract languages for security (e.g. [8]) do not usually address information flow, putting the stress instead on access control. Regarding enforcement of information flow, there are certainly static solutions; but, in fact, we are not aware of any that use runtime methods. The static approach usually comes in the shape of a type-system to enforce *noninterference* [50], where the idea is to prevent all flow of information from the domain of secrets to the public-domain. It has been noted however that noninterference is unsuitable in most real-life situations. There, an application is expected to declassify some well-defined piece of information, thus creating the need to admit some flows of secret information to the public-domain. Type systems that try to accommodate declassification, e.g. [43], soon suffer from the so-called *label creeping problem*: A security type system, which associates a classification (or security label) to each piece of data, necessarily describes an abstraction of a set of values, possibly losing precision every time the value participates in a computation. The accumulation of these losses results in type systems that, in order to remain secure, reject too many secure systems [19].

On the other side, it is well-known that information flow properties are actually not safety properties (in fact, they do not even qualify as *properties* in the Alpern-Schneider classification [5]). Therefore, runtime approaches are generally considered inappropriate, since they are naturally associated with the enforcement of safety properties.

Recent results by Hamlen et al. [42] and by Ligatti et al. [36] hint at the potential of code rewriting techniques as a framework to accommodate several enforcement mechanisms. There is a profusion of work on code rewriting techniques (see [57, 56] for two thorough surveys) with applications ranging from compilation, program synthesis and optimization to refactoring and reverse engineering. However, not much research has been devoted to study code rewriting for policy enforcement. A remarkable exception is [42] where it is shown that RW-enforceable policies (i.e. policies enforceable using code rewriting) strictly include those enforceable using reference monitors and/or static analysis. These results provide strong evidence that approximations of information flow properties may be RW-enforceable, i.e. policies that can be enforced using code rewriting, cf. the "Secret File Policy" example [42] and [29].

3 Programming languages and SOA

Current programming language abstractions are not good enough 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 [39] Besides, many current production OOP languages (e.g. Java and C#) are based on the shared-state model of concurrency so they do not handle concurrency and message passing particularly well. Another criticism to OOP concerns the possibility of reusability. Object-orientation provides two distinct mechanisms for composing concerns: aggregation and inheritance. Some examples show [4] that reusing components through aggregation and inheritance mechanisms may not be successful when the objects implement concerns like history information, multiple views and synchronization. OOP needs therefore better abstraction mechanisms.

The Creol project [18] has been addressing many of the objections to object-orientation. Essentially, a Creol program consists of concurrent objects communicating asynchronously and with internal process control. By means of mechanisms for conditional processor release points, passive waiting, and time-out [33, 34], explicit synchronization primitives are not needed in the language. An abstract representation of the Creol architecture is shown in Fig. 2. Compared to for instance Polyphonic C#, Creol has a simpler set of communication primitives using the concept of asynchronous method call. By staying within the method paradigm, inheritance and overloading is unproblematic. Creol allows multiple inheritance, which is not supported by Java, Polyphonic C#, nor join calculus based languages. Instead of the standard AOP mechanisms, which hinder program reasoning, Creol offers a synchronized merge operator which may be seen as a high-

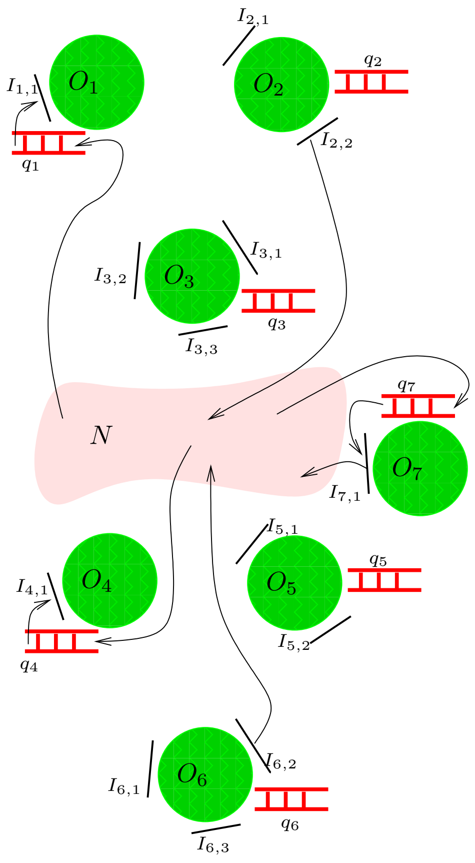

Figure 2: The Creol Architecture. For each object O_i: $I_{i,j}$ are its interfaces and q_i its message queue. N is the network.

level AOP-like construct, and effectively reduces the problems related to the so-called inheritance anomaly [37], while allowing reasoning.

XML documents are not yet integrated in the Creol language, however, one may easily model an abstraction of XML documents in Creol, using Creol's data types, which includes inductively defined data types and a functional sub-language (similar to, for instance, Haskell). Since all messages and immutable values are defined by data types in Creol, it is not natural to define XML documents by the class mechanism, as would be the option in most other object-oriented languages.

4 Research directions

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

- Formal definition of generic contracts. Currently, there is no unified formal definition of contracts (in particular for QoS and confidentiality).

- 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.

- Language-based support for contracts. In the literature (e.g., [39]) it has been identified that the following three areas must have a language-based support: (a) data-access, (b) concurrency and (c) security. A fourth area has to be considered: (d) contracts; currently, 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. There is a big mismatch 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 proving/guaranteeing essential contract properties. Guaranteeing the non-violation of contracts might be done in (at least) four different ways: 1. enforcement at runtime, through monitors, for instance; 2. by construction, e.g. through low-level language mechanisms; 3. static analysis withstandard program analysis techniques; or 4. model checking. None of the above can be used as a generic, universal tool for inferring all the properties of contracts. Different approaches must be used for different properties.

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 confidentiality. A minimum requirement is the ability to seamlessly combine real-time models (for QoS specification) and behavioral models (essential to constrain protocol implementation and to enforce confidentiality). Contract models should also address discovery and negotiation. Regarding confidentiality, it seems that more experiments with RW-enforceable policies giving sufficient conditions for *admissible information flow* [30] can be envisaged. The objective should be to develop practical and efficient methods to enforce information flow properties of realistic code, including cryptographic protocol implementations.

Yet, the formal definition of contracts should be only a first step towards a more ambitious task, namely a language-based support for programming and effectively use 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 could be interesting to investigate a language primitive to create wrapped objects which are correct-by-construction. Firewalls may then be implemented in this way. On the other hand, contracts for QoS and confidentiality 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 clearly exposed in the ARTIST road-map [15], finding languages or notations for describing timing behaviors and timing requirements is easy; the real challenges are in analysis, i.e. to check that the requirements are guaranteed. So, besides the syntactic extensions mentioned above, the language needs to have timing semantic extensions in order to allow extraction of a timed model, e.g. a timed automaton. It may be checked with existing tools e.g., Kronos [61] and Uppaal [11]. Model checking tools will help to prove real-time properties, like guaranteeing that a given promise service will, for instance, satisfy it response-time constraint. Other properties may, instead, be proved to be correct-by-construction (e.g. wrappers, as mentioned above).

In practice, many properties cannot be proved correct using correct-by-construction or model checking techniques. In such cases only a runtime approach may be used. It seems that a promising direction is to develop techniques for constructing runtime monitors from contracts. In this case, monitors will be used to enforce the non-violation of contracts.

5 A specific proposal

We believe object-orientation is still a good paradigm for modeling open distributed systems. The main problems with object-orientation come from language design and implementation decisions, not from its original philosophy. The Creol project has addressed many of these problems. Creol has a formal semantics defined in rewriting logic [40] and implemented in Maude [38], and supports compositional program reasoning. In addition, the dynamic class

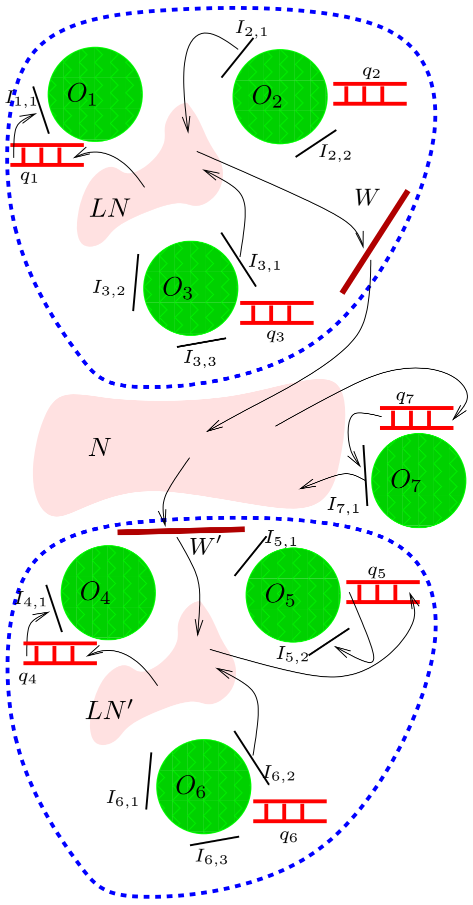

Figure 3: The Extended Creol Architecture

construct of Creol is well suited for dynamic reconfiguration and maintenance of services in large networks. In its current state, Creol has basic constructs that are suitable for programming the Internet in an object-oriented manner. Since its operational semantics is executable in the Maude tool, a language interpreter is readily available. In addition, the various Maude commands for model checking and exhaustive search are available for Creol programs.

By using Creol and its definition in rewriting logic as our framework, we propose the following:

- Formalization of contracts (for confidentiality and QoS) using a timing extension of rewriting logic.
- Use of the meta-level capabilities of rewriting logic to specify contract negotiation protocols.
- Syntactic extension of Creol to include contracts as interfaces.

- Integration of XML in Creol.
- Syntactic and semantic extension of Creol aiming at extracting timed models amenable to model checking.
- Analysis of the timed models using current model checking tools.
- Runtime monitoring of contracts.

Below we explain in more details the items above.

Regarding the formal definition of contracts, many formalisms may be used, but we believe such a generic model can be described harmoniously using real-time extensions of rewriting logic [62]. This is in line with recent investigations in the use of rule languages to model contracts [32, 45]. While these rule-based languages are essentially ad-hoc, we expect to profit from the existing large body of research in rewriting logics.

The rule-based approach promoted by the research mentioned above brings along new challenges in the definition of appropriate negotiation schemes [49, 9, 47]. Here again, rewriting logic can give invaluable help. Its reflection and meta-level computation properties may help define and structure the negotiation protocol.

After defining contracts with suitable negotiation protocols in a solid formal theory, we would like to concentrate on Creol extensions. By defining interfaces on components consisting of a collection of objects, we develop a notion of contract for such interfaces that integrates the main expressive power of composition filters. In addition, the implementation of rewriting logic by the Maude tool enables rapid prototyping and evaluation of alternative designs, which is essential for finding practically useful solutions. The analysis tools of Maude will be valuable when assessing their properties. The interface concept of Creol is oriented towards specification of observable behavior, expressed by means of the interaction history, i.e. the sequence of all (visible) messages to or from an object.

A full integration of XML documents in Creol would require an extension of the language. In particular, the use of regular expressions should be integrated in the functional sub-language, to allow flexible retrieval.

When adding real-time, Creol interfaces may be used to specify static and dynamic contracts. Furthermore, semantics extensions of Creol are needed in order to extract a timed automaton amenable to be model checked.

Another interesting extension of Creol would be to augment the interface syntax with mechanisms for specifying dynamic contract monitoring. Moreover, the executable operational semantics of Creol could be used to test the approach in situations where formal verification is practically impossible (e.g., confidentiality properties). Additionally, the meta level of Maude may well be used for monitoring without affecting the application code.

The proposed extended Creol architecture is shown in Fig. 3. Comparing with Fig. 2, the extension consists of *wrappers* enveloping sets of objects, possibly of different classes and communicating through their own local networks (LN and LN'). The access from outside the wrapper will be regulated by the wrapper interface W. Contracts will be defined both at local (object) interfaces as well as at wrapper interfaces.

6 Conclusion

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. Software Engineering deals with the first aspect while the second one is concerned with models of computation involving expressiveness results, verification and security [41].

Moreover, in order to make collaboration a reality among different web-services, the formal definition of monitorable and negotiable contracts has become an imperative.

In this paper we have surveyed main current approaches to program web-services and the features of state-of-the-art programming languages used. We have identified some problems and open issues of current approaches (see Section 4) and we have proposed general research directions and a particular road-map based on Creol (Section 5).

The next natural step is to map the expected results into real languages. One possibility would be to translate Creol programs into existing web-services languages. However, this approach does not seem realistic, mainly because the currently available target languages are far from being suitable for such ambitious task. In our opinion the right approach would be to develop a contract-based language from scratch, capitalizing on the Creol experience.

References

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

[2] M. Aksit, L. Bergmans, and S. Vural. An object-oriented language-database integration model: The composition-filters approach. In *ECOOP '92: Proceedings of the European Conference on Object-Oriented Programming*, pages 372–395, London, UK, 1992. Springer-Verlag.

[3] M. Aksit, J. Bosch, W. van der Sterren, and L. Bergmans. Real-time specification inheritance anomalies and real-time filters. *Lecture Notes in Computer Science*, 821:386–??, 1994.

[4] M. Aksit and B. Tekinerdogan. Solving the modeling problems of object-oriented languages by composing multiple aspects using composition filters, 1998.

[5] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, Oct. 1985.

[6] J. M. Andreoli and S. Castellani. Towards a Flexible Middleware Negotiation Facility for Distributed Components. In *DEXA '01: 12th International Workshop on Database and Expert Systems Applications*, page 732. IEEE Computer Society, 2001.

[7] I. Assayad, V. Bertin, F.-X. Defaut, P. Gerner, O. Quevreux, and S. Yovine. Jahuel: A formal framework for software synthesis. In *ICFEM*, LNC, 2005. To appear.

[8] J. S. B. de Win, F. Piessens and W. Joosen. Towards a Unifying View on Security Contracts. In *Software Engineering for Secure Systems – Building Trustworthy Applications (SESS'05)*. ACM, 2005.

[9] C. Bartolini, C. Preist, and N. R. Jennings. A Generic Software Framework for Automated Negotiation. Technical Report HPL-2002-2, HP Laboratories Bristol, Jan. 2002.

[10] C. Becker and K. Geihs. Quality of Service and Object-Oriented Middleware-Multiple Concerns and their Separation. In *21st International Conference on Distributed Computing Systems Workshops (ICDCSW '01)*, 2001.

[11] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in LNCS, pages 232–243. Springer–Verlag, October 1995.

[12] L. Bergmans and M. Aksit. Composing crosscutting concerns using composition filters. *Commun. ACM*, 44(10):51–57, 2001.

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

[14] G. Bierman, E. Meijer, and W. Schulte. The essence of data access in Cω. In *European Conference on Object-Oriented Programming*, 2005.

[15] B. Bouyssounouse and J. Sifakis, editors. *Embedded System Design: The ARTIST Roadmap for Research and Development*, volume 3436 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

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

[17] D. Cooney, M. Dumas, and P. Roe. A programming language for web service development. In *CRPIT '38: Proceedings of the Twenty-eighth Australasian conference on Computer Science*, pages 143–150, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[18] Creol Homepage. `http://www.ifi.uio.no/~creol/`.

[19] M. Dam and P. Giambiagi. SPC 01-4025 Mobile Language Study, Final Technical Report. Technical report, EOARD, 2003. `http://www.sics.se/~pgiamb/Publications/eoard-TR2003.ps.gz`.

[20] A. Daskalopulu. Model Checking Contractual Protocols. In L. Breuker and Winkels, editors, *Legal Knowledge and Information Systems, JURIX 2000: The 13th Annual Conference*, Frontiers in Artificial Intelligence and Applications Series, pages 35–47. IOS Press, 2000.

[21] A. Daskalopulu and T. S. E. Maibaum. Towards Electronic Contract Performance. In *Legal Information Systems Applications, 12th International Conference and Workshop on Database and Expert Systems Applications*, pages 771–777. IEEE C.S. Press, 2001.

[22] H. Davulcu, M. Kifer, and I. V. Ramakrishnan. CTR-S: A Logic for Specifying Contracts in Semantic Web Services. In *Proceedings of WWW2004*, pages 144–153, May 2004.

[23] G. Diaz, J.-J. Pardo, M. E. Cambronero, V. Valero, and F. Cuartero. Automatic translation of WS-CDL choreografies to timed automata. In

Proceedings of 2nd International Workshop on Web Services and Formal Methods (WS-FM 2005), September 2005.

[24] G. Diaz, J.-J. Pardo, M. E. Cambronero, V. Valero, and F. Cuartero. Verification of web services with timed automata. In *Proceedings of 1st International Workshop on Automated Specification And Verification of Web*, March 2005.

[25] ebXML: Electronic Business using eXtensible Markup Language. `www.ebxml.org`.

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

[27] D. Florescu, A. Grünhagen, and D. Kossman. XL: An XML programming language for web service specification and composition. In *Proc. The Eleventh Int'l World Wide Web Conference*, pages 65–76, May 2002.

[28] D. Florescu, A. Grünhagen, and D. Kossman. XL: A platform for Web services. In *Conference on Innovative Data Systems Research (CIDR)*, 2003.

[29] P. Giambiagi. *Controlled Declassification of Information*. PhD thesis, Royal Technical University, Stockholm, Sweden, In preparation 2005.

[30] P. Giambiagi and M. Dam. On the Secure Implementation of Security Protocols. *Science of Computing Programming*, 50:73–99, 2004.

[31] G. Governatori. Representing business contracts in RuleML. *International Journal of Cooperative Information Systems*, 14:181–216, 2005.

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

[33] E. B. Johnsen and O. Owe. An asynchronous communication model for distributed concurrent objects. In *Proc. 2nd Intl. Conf. on Software Engineering and Formal Methods (SEFM'04)*, pages 188–197. IEEE Computer Society Press, Sept. 2004.

[34] E. B. Johnsen and O. Owe. Object-oriented specification and open distributed systems. In O. Owe, S. Krogdahl, and T. Lyche, editors, *From Object-Orientation to Formal Methods: Essays in Memory of Ole-Johan Dahl*, volume 2635 of *Lecture Notes in Computer Science*, pages 137–164. Springer-Verlag, 2004.

[35] S. Kraus. Automated Negotiation and Decision Making in Multiagent Environments. *Lecture Notes in Artificial Intelligence*, 2086:150, 2001.

[36] J. Ligatti, L. Bauer, and D. Walker. Edit automata: Enforcement mechanisms for run-time security policies. *International Journal of Information Security*, 4(1–2):2–16, Feb. 2005.

[37] S. Matsuoka and A. Yonezawa. Analysis of inheritance anomaly in object-oriented concurrent programming languages. *Research directions in concurrent object-oriented programming*, pages 107–150, 1993.

[38] Maude System. `http://maude.cs.uiuc.edu/`.

[39] E. Meijer, W. Schulte, and G. Bierman. Programming with circles, triangles and rectangles. In *Proceedings of the XML Conference*, 2003.

[40] J. Meseguer. Research directions in rewriting logic. In *Computational Logic*, volume 165 of *Lecture Notes in Computer Science*, Marktoberdorf, Germany, 1997. NATO Advanced Study Institute, Springer-Verlag.

[41] U. Montanari. Web services and models of computation. In *First International Workshop on Web Services and Formal Methods*, volume 105 of *Electronic Notes in Computer Science*. Elsevier, 2004.

[42] G. Morrisett, F. B. Schneider, and K. Hamlen. Computability classes for enforcement mechanisms. Technical Report 2003-1908, Cornell, 2003.

[43] A. C. Myers. JFlow: Practical mostly-static information flow control. In *Proceedings of the 26th POPL*, pages 228–241, San Antonio, TX, Jan. 1999. ACM.

[44] M. P. Papazoglou. Service-Oriented Computing: Concepts, Characteristics and Directions. In *4th International Conference on Web Information Systems Engineering (WISE)*. IEEE CS, 2003.

[45] A. Paschke, M. Bichler, and J. Dietrich. ContractLog: An Approach to Rule Based Monitoring and Execution of Service Level Agreements. In *RuleML*, 2005.

[46] A. Paschke, J. Dietrich, and K. Kuhla. A Logic Based SLA Management Framework. In *4th Semantic Web Conference (ISWC 2005)*, 2005.

[47] A. Paschke, C. Kiss, and S. Al-Hunaty. A Pattern Language for Decentralized Coordination and Negotiation Protocols. In *EEE*, pages 404–407, 2005.

[48] W. Picard. NeSSy: Enabling Mass E-Negotiations of Complex Contracts. In *DEXA '03: 14th International Workshop on Database and Expert Systems Applications*, page 829. IEEE Computer Society, 2003.

[49] D. M. Reeves, M. P. Wellman, and B. N. Grosof. Automated negotiation from declarative contract descriptions. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 51–58. ACM Press, 2001.

[50] A. Sabelfeld and D. Sands. A PER model of secure information flow in sequential programs. *Higher-Order and Symbolic Computation*, 14(1), 2001.

[51] Simple Object Access Protocol (SOAP). `http://www.w3.org/TR/soap/`.

[52] C. Szyperski. Component technology - what, where, and how? In *Proceedings of the 25th International Conference on Software Engineering (ICSE)*, pages 684–693. IEEE, 2003.

[53] A. Tesanovic, D. Nyström, J. Hansson, and C. Norström. Aspects and components in real-time system development: Towards reconfigurable and reusable software. *Journal of Embedded Computing*, 1(1), 2 2004.

[54] V. Tosic. On Comprehensive Contractual Descriptions of Web Services. In *IEEE International Conference on e-Technology, e-Commerce, and e-Service*, pages 444–449. IEEE, 2005.

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

[56] E. Visser. A survey of rewriting strategies in program transformation systems. *Electronic Notes in Theoretical Computer Science*, 57, 2001.

[57] E. Visser. A survey of strategies in rule-based program transformation systems, March 2004. (Draft).

[58] WSA. Web Services Architecture. W3C Working Group Note, `www.w3.org/TR/ws-arch/`, Feb 2004.

[59] Web Services Agreement Specification (WS-Agreement). `https://forge.gridforum.org/projects/graap-wg/document/WS-AgreementSpecification/en/7`.

[60] WSLA: Web Service Level Agreements. `www.research.ibm.com/wsla/`.

[61] S. Yovine. Kronos: A verification tool for real-time systems. *International Journal of Software Tools for Technology Transfer*, 1(1/2):123–133, October 1997.

[62] P. Ölveczky. Specification of real-time and hybrid systems in rewriting logic. *Theoretical Computer Science*, 285, 2002.