Formal Methods for Component Description, Coordination and Adaptation

Pascal Poizat[1], Jean-Claude Royer[2], and Gwen Salaün[3]*

[1] LaMI, UMR 8042 - CNRS, Université d'Évry Val d'Essonne, Genopole
[2] OBASCO Project - École des Mines de Nantes, INRIA
[3] DIS, Universita di Roma "La Sapienza"

Abstract. Components, connectors and architectures have now made a breakthrough in software industry, leading to Component-Based Software Engineering (CBSE). In this paper, we argue for the pragmatic use of formal methods to design and reason on CBSE systems at an abstract level and to solve CBSE issues. We give some possible benefits of such an approach and list some of its open issues.

1 Introduction

With the increase of complexity of software systems in the last few years, the Component-Based Software Engineering (CBSE) approach emerged as a discipline which yields promising results such as trusted and/or On-The-Shelf components (COTS), increase of the components reusability, semi- or automatic composition and adaptation of components into architectures, expressive middlewares and so on. A major drawback of the mainstream approach for CBSE is that it mainly focused on programming/low level features making it difficult to reason on problematic issues hidden behind the CBSE promised results. Moreover, with the increasing number of different implementation-level CBSE languages/frameworks all had to be studied (and learned[1]) over and over without regards to the real abstract concepts and issues hidden behind.

Recent meta-descriptive approaches such as the OMG MDA and AOP/AOSD promote the return to abstract models and clear separation between the functional (*i.e.* business, abstract) and the technical or implementation aspects of systems (such as real-time constraints or synchronizing policies). Orthogonally, Architectural Description Languages [34] and Coordination Languages [39] are also increasingly promoted in the CBSE community. Both are interesting with regards to their ability to abstract the systems and provide better/simpler reasoning mechanisms on them, but also because this activity can be formally grounded and then equipped with tools which do not limit themselves to be drawing ones.

* G. Salaün's work is partially supported by Project ASTRO funded by the Italian Ministry for Research under the FIRB framework (funds for basic research).

[1] This in some way saves the book industry ...

In this position paper our goal is to advocate for the use of ADL and (abstract) coordination languages to design and reason on CBSE and for the application of formal techniques to complement both.

2 Position: Abstraction and Formalism

2.1 The Need for Abstraction

The huge number of platforms for components and/or reactive objects, either industrial ones (*e.g.* EJB, CCM, Web Services) or more academic ones (*e.g.* Fractal, ArchJava, JAC, SugarCubes) has a cost: teaching issues (which one to learn?), maintenance problems in the enterprise, interoperability problems, ... As a solution, the OMG advocates for its Model Driven Approach (MDA) in which models are at the core of the engineering process [9], abstracting away (at least in a first step) from implementation details. ADLs and Coordination languages (CLs) are, in our opinion, good candidates to describe such models. Here we address both, as both are closely related. The main difference between them is that CLs require a clear separation between coordination [24] and computation (that is component internals which are not taken into account). This principle is not always ensured by ADLs (hence not all ADLs are CL candidates). Both ADLs and CLs deal principally with component interfaces and glues (or connectors, which may or not be components). Note that this is not true for Linda-like tuplespace based CLs. While this could be a very expressive framework to implement coordination mechanisms (see for example the implementation of tuplespaces in Java, JavaSpaces), we think that it is too closely related to implementation issues, making it difficult to formally verify coordinators in a compositional way. Coordination languages should be in our opinion more abstract and mainly focused around the concept of events or interactions between coordinated entities which seems better suited to deal with interoperating issues. However, the remaining of this paper may also stand for Linda-like languages (see for example works on process algebras used to compare expressiveness between coordination languages in [14]).

2.2 What to Formalize?

The first important issue with components is related to the definition of their *interface* using Interface Description Languages (IDL). This has been addressed by CBSE frameworks first to be able to statically (*i.e.* at compile-time) generate skeletons and stubs for the communicating components (beginning by CORBA and RMI). More recently, this issue has found itself at the core of the most challenging issues dealing with the last promising component framework, Web Services, to deal with composition and choreography issues (see Section 3).

The limit of signature types IDLs has now been demonstrated. Type correct communicating components may deadlock because they do not have compatible protocols. It is therefore now widely accepted that IDLs have to take into

account such *behavioural protocols*. This may be used either as a piece of documentation for the components, in a design-by-contract process (*e.g.* Trusted Components), to compose and check connections between components or even build adapters [52] when component interfaces do not match.

We also have to model expressive means to *glue* the components, that is to express the semantics of putting several communicating components altogether.

Both IDLs and glues have the need for some description of datatypes. In IDLs, datatypes describe the types exchanged between components. In glues, datatypes are used to express value-passing, synchronizing constraints or components identifiers.

The need for the modeling of components themselves only appears in the ADL framework. A simple way is to be consistent with component IDLs and express components behaviours in the same way than we model their behavioural abstraction.

2.3 Which Formal Language?

In our more recent works [3, 1, 42, 46] we advocate for the joint use of three formal languages:

- a glue language (`GL`), gluing components altogether, describing explicit connectors (in ADLs) and coordination patterns (in CLs). This language has to be very abstract as it should be possible to use it alone. Modal logics are the best candidates for this, yielding a good compromise between expressiveness and abstraction.
- a static description language (`SL`) to deal with the static parts of the system (datatypes). These datatypes must be as abstract as possible, *i.e.* mainly a type name, operation profiles and some properties for these operations. We think that algebraic datatypes or model-oriented languages (Z, B) are good candidates for `SL`.
- a dynamic description language (`DL`) to deal with the dynamic parts of the system (behavioural interfaces or components behaviours in ADLs). Several languages are possible (Petri nets, state/transition diagrams, process algebras, temporal logics), but the integration of `SL` into `DL` and `GL` has to be simple. Petri nets are not compositional and quite difficult to verify in presence of data (High level Petri nets), while this has been resolved for value-passing process algebras and transition systems using Symbolic Transition Systems [28, 16]. Process algebras are more expressive and transition systems more readable, but only process algebras have the extension power to be able to tackle so many different aspects such as time (TCSP), different synchronizing or concurrency policies (SCCS, ACCS, TACCS, CSP, LOTOS), probabilities/stochastics (EMPA, PEPA) or locality and mobility (π-calculus and its dialects) into account. Hence we think `DL` should be a process algebra.

2.4 Which Benefits?

The main benefits of using formal methods in an approach as described above are the following.

Animation. Process algebras always have a structural operational semantics (SOS) defined. Using a SOS, one may develop animating tools for prototyping matters. These tools are numerous for simple process algebras (*e.g.* CWB-NC which deals with both CCS, CSP, SCCS and TCSP, Mobility Workbench for π-calculus, ...[2]). However, as far as value-passing behavioural languages are concerned, only some tools do exist. CADP [25] is a powerful tool dedicated to LOTOS, *i.e.* dealing only with algebraic specifications for `SL`. LTSA [31] is dedicated to the FSP process algebra which underlies several formal proposals for component composition and analysis (the last one is [19]). xCLAP [4] is a prototype animating tool for our [3] approach in which `SL` may be either algebraic specifications, Z or B.

Equivalences, preorders and compositionality. Equivalences in the process algebraic framework[3] are numerous [51]. They are more or less coarse and one may use the one or the other depending on the kind a abstraction he/she wants for its component IDL (full abstraction). Equivalences enable of course one to check whether two components are equivalent but they may also used to check if a component is compliant with reference to its IDL protocol or if architectural bindings are consistent as in the Wright ADL [2]. Related preorders enable to check component or connector refinement. The equivalences commonly used in the formal CBSE framework are:

- bisimulation (weak and strong versions). The weak one is the more adequate as it may be used to check if a hight level component protocol and a composition, hiding internal communications, are equivalent (this has been applied to Web Services for example, see Section 3). Weak bisimulation is not a congruence however a congruent version (observational equivalence) may be obtained.
- trace equivalence relates the traces of two components. It is related to may testing equivalence which yields for two processes whenever they may pass the same set of tests.
- failures/divergences equivalences, acceptance/refusal sets equivalences, and must testing, are equivalences that usually correspond to the compliance idea but taking some form of refinement into account (refined components may provide new services for example). These equivalences have been used a lot (under different forms) in the formal CBSE framework, *e.g.* [38, 48, 18, 40, 2].

[2] A more comprehensive list of tools can be found at `http://www.lami.univ-evry.fr/~poizat/liens-en.php`.

[3] Note that these equivalences are defined on the models of process algebras, *i.e.* LTS, hence they can also be used on transition diagrams behavioural languages.

Compositionality is a common property in the process algebraic framework. There are different characterizations of it, the more interesting one states that, given some equivalence $=$ between processes, if two processes p and q are such that $p = q$ then, for any context C (which corresponds in our actual domain of interest to some architectural composition), we have $C[p] = C[q]$. Compositionality is ensured when the equivalence is a congruence. This property enables one to replace a given component (resp. connector) by a proven equivalent component (resp. connector) in a configuration and obtain a configuration which is equivalent. Note that compositionality may be used to replace components or connectors by compositions. Compositionality also exists in a preorder version.

Deadlock freedom and temporal properties. Deadlock freedom can be checked to see if a composition of components and connectors is correct (*e.g.* [2, 8]). Liveness properties (and more generally properties expressed in temporal logics) can be checked using model checking techniques. High level properties of components or connectors can be expressed in such a logic and checked against a given implementation of it. These techniques have already proven useful to check real-life component models [49, 35].

Adapters. When components and connectors, or their ports/roles have a formal semantics then one may go beyond checking name/static type correspondence and use this semantics to build out adapters which may correct interface inconsistencies. There are few formally grounded pieces of work on this [50, 12] since the seminal work of [52]. They enable one to reason on adapters (*i.e.* address preorder or compositionality issues), but these approaches still require the manual design of adapters which may not be automatically derived from the component interfaces (this would somehow require an ontology to know what is behind names of required/provided services).

Specific aspects. All the different benefits given above could be adapted to a specific aspect process algebra (time, synchronicity, stochastics, ...). Examples of such aspect specific `DL`s are π-calculus (and its dialects) used to express topologic aspects of architectures (*e.g.* Darwin [43]) or performance properties which may be checked against stochastic process algebras (*e.g.* Æmilia [5]). Corresponding `GL`s could be respectively the named μ-calculus used in the Mobility Workbench or the spatial logic of [15], and stochastic temporal logics. However, it is not yet sure that a `GL` does exist for any possible aspect `DL`. Very recently this process algebraic way of designing and checking components has even been applied to DNA and protein matters in biological regulatory networks (BioSPI).

In the next Section, we come back on some of these issues on maybe the most challenging domain: Web Services (indeed, the CBSE Framework where more and more formal methods applications are being experimented).

3 A Specific Application Domain: Web Services

Web Services (WSs for short) emerged recently and are a promising way to develop applications through the internet (their main specificity compared to more traditional software components). WSs are distributed, independent pieces of code (people often say *processes* due to their dynamic foundation) which communicate with each other through message exchange. Therefore, the central question in WS engineering is to make a number of processes work together to perform a given task.

Executable applications are supported by existing XML-based technologies: WSDL interfaces abstractly describe messages to be exchanged, SOAP is a protocol for exchanging structured information, UDDI is a repository to publish and discover WSs, BPEL4WS is a notation for specifying business process behaviours. Another alternative to describe the latter is the DAML-S (recently renamed OWL-S) proposal funded on ontologies. Some industrial platforms already partially implement some of these standards, especially Microsoft's .NET and Sun's J2EE.

WSs raise many theoretical and practical issues which are part of on-going research. Some well-known open problems related to WSs are to specify them in an adequate, formally defined and expressive enough language, to compose them (automatically), to discover them through the web, to ensure their correctness, etc. In this section, we argue that the use of formal methods is valuable as an abstract way to deal with WSs and then to tackle several issues. We especially focus herein on the description, coordination and adaptation questions.

At this abstract level, lots of proposals tended to describe WSs using either semi-formal notations (no formal meaning therefore error-prone and not supported by development tools, except for edition means), especially workflows [30], or some more formal proposals grounded for most of them on transition system models (Labelled Transition Systems, Mealy automata, Petri nets) [27, 37, 26, 7, 29]. Composition is the fact to find the right way to put together a judicious bundle of WSs to solve a precise task. On the other hand, orchestration is more dedicated to monitoring WSs involved in a composition for example. Choreography is the issue to find out a judicious expressiveness level for WS public interfaces (possibly used for composition purpose afterwards). As far as the composition issue is concerned, different techniques have emerged which ensure a correct composition such as automatic composition [7, 33], planning [32, 29] or model deduction [37]. With regards to the reasoning issue, works have been dedicated to verifying WS description to ensure some properties of systems [20, 37, 19, 36]. Summarizing these works, they use model checking to verify some properties of cooperating web services described using XML-based languages (DAML-S, WSFL, WSDL, BPEL4WS, WSCI). Accordingly, they abstract their representation and ensure some properties using ad-hoc or more well-known tools (*e.g.* SPIN, LTSA).

In this context, a recent trend aim at advocating the use of process algebra in the WS world [13, 44, 45]. This approach can be viewed as an alternative proposal. Compared to the previous works, process algebra are adequate to describe

web services, because they are formal, based on simple operators but expressive enough, and equipped with tools to support the design and ensure temporal properties. Additionally, their constructs are adequate to specify composition due to their compositionality property. At the communication level, it is worth noting that several other dynamic techniques could be used to coordinate dynamic entities and particularly WSs, such as synchronized products, interactions diagrams, semantic glue, temporal logic, etc (see [46] for supplementary explanations).

4 Open Issues

Open issues are twofold. On the one hand, formal methods will not be integrated into the software development and reuse process if some requirements are not fulfilled:

- graphical counterparts for formal/textual notations have to be developed, when possible. If such graphical notations do exist for the architectural description part of the systems (see AcmeStudio [47] for example), expressive (and explicit) gluing mechanisms are not taken into account. However, the formal language we proposed for mixed specifications, Korrigan [41, 42], has a graphical notation for temporal logic glues [17].
- seamless process for the integration of formal methods into a non-formal CB one. Tools and methodologies are needed here. In the ADL framework, languages such as Acme [22] and its tool, AcmeStudio, have already been used to build a verification process involving several less abstract ADLs (Wright, Rapide and AESOP, see [23]). However the main problem is to translate the results of the verification process (error traces, characterizing temporal formulas, ...) back into the non-formal language (MSCs, animations into ADL tools, ...)
- account into formal IDLs/glues for any needed aspect. A question here is: which aspects are needed? (types, behaviours, time constraints, probabilities, synchronizing policies, ...)

On the other hand, some more theoretical issues have yet to be solved:

- if formal methods[4] are available to model any aspect of CBSE, none really takes into account all aspects one would want to model. Building mixed languages (*i.e.* taking several aspects into account) is a solution (see [41] for a list of languages dealing with the static+dynamic aspects for example). However taking lots of aspects into account leads to complex theoretical issues. For example, all the process algebraic tools for verification (bisimulation, model-checking adequate logics, ...) had to be adapted in the value-passing case in order to avoid state explosion problems [28, 16]. Adaptation had also to be achieved for asynchronous communication (*e.g.*, testing equivalences [11]). Moreover, the more complex is the formal description language, the more

[4] This is true even if one restricts itself to the process algebraic framework.

huge is the complexity of verifications using it (if decidable!). An interesting solution is to apply views/aspect-oriented techniques to formal languages. Here again, some specific solutions have been developed (see for example [10] on LOTOS + Z) but there is not really a general/abstract framework for formal languages weaving. We are currently investigating this issue at the more abstract level, working on the weaving[5] of logics such as for example the one we specifically built to address refinement issues in mixed ADLs [1]. The interest of working at the logic level is that abstract results on weaved logics (compositionality, reuse/composition of model-checking techniques or equivalences, ...) could be thereafter applied to instantiations of these logics, *i.e.* formal languages dedicated to different aspects. In a certain sense, this correspond to the OMG 4-level meta-modeling architecture, with logics beings a level meta-meta (M3) and languages at level meta (M2). The difference is that we use logics (which are are the core of any formal(ized) technique and not the MOF).

- complexity issues. State explosion problems or explosion in time are well-known problems of the application of formal methods to real-size systems. Even if abstract descriptions are advocated in the ADL, and even more in the coordination framework, complexity issues are still raised by genericity on types, on number of components in architectures, ... Process algebras, their underlying compositionality principles, and value-passing extensions, are ways to tackle these issues. However, dealing with real applications (and not models, specifications) there are no more nice levels of abstractions. In the testing framework, we are actually working within the context of a national research funded project on the definition of abstract and compositional testing techniques for components. Our goal here is to be able either to (i) reuse tests made on subcomponents when testing compositions using them, or (ii) derive the minimal set of symbolic tests to be made on subcomponents from the test-requirements for the whole system and the (formal) knowledge we have on the connections used between them (*i.e.* the middleware). This leads us to another issue:
- what is a good level of abstraction for an IDL? This relates to the definition of equivalences fully abstract with reference to some specific need of the designer, and which may then lower the complexity level of formal verification.
- does a `GL` exist for each possible aspect `DL` process algebra? Moreover, is really using modal/temporal logics as `GL`s the best (more expressive) solution? Possible alternatives are traces specifications or patterns [2, 21, 6].

More particularly to the WS domain:

- Compatibility and adaptation are a worthy open issue in this domain. The matter is to assess the compatibility of two services taking into account their public interfaces, and consequently to make it possible to compose two (or more) WSs. In case of incompatibility, one has to use adapters to ensure a correct matching of the involved WSs and of their public interfaces (it

[5] The term used is *fibring* in the logical framework.

depends then on the way these latter are described). A related issue is the substitutability question. The goal is to substitute a service by another one in a given context while preserving the meaning of the whole. An idea we come up with is the use of preorder and bisimulation notions (particularly weak bisimulation, congruence and symbolic bisimulation) to allow substitutions of WSs modulo such equivalences.

5 Conclusions

The CBSE approach yields promising results such as the increase of reusability level of software entities or the automatic adaptation and composition of components. However, these results raise difficult issues to be solved first. Languages/techniques such as ADL or coordination languages may then be of great use as they may, following the MDA main proposal, abstract away from platform specificities and reason on the core (business, functional) models. We think that formal methods, used pragmatically, can be of great interest in the ADL and coordination languages frameworks, enabling one to reason on difficult issues for CBSE systems.

References

1. M. Aiguier, F. Barbier, and P. Poizat. A Logic with Temporal Glue for Mixed Specifications. In *Proc. of the 2nd Int. Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'03)*, France, 2003. To appear in ENTCS.
2. R. Allen and D. Garlan. A Formal Basis for Architectural Connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–249, 1997.
3. C. Attiobé, P. Poizat, and G. Salaün. Integration of Formal Datatypes within State Diagrams. In Mauro Pezzè, editor, *Proc of the 9th Int. Conf. on Fundamental Approaches to Software Engineering (FASE'03)*, volume 2621 of *Lecture Notes in Computer Science*, pages 341–355, Poland, 2003. Springer-Verlag.
4. C. Attiogbé, A. Auverlot, C. Cailler, M. Coriton, V. Gruet, M. Noël, and G. Salaün. The xCLAP Toolkit. Available at `http://www.dis.uniroma1.it/~salaun/xCLAP/`, 2003.
5. S. Balsamo, M. Bernardo, and M. Simeoni. Combining Stochastic Process Algebras and Queueing Networks for Software Architecture Analysis. In *Proc. of the Int. Workshop on Software and Performance (WOSP'2002)*, 2002.
6. G. Batt, D. Bargamini, H. de Jong, H. Garavel, and R. Mateescu. Model Checking Genetic Regulatory Networks using GNA and CADP. In S. Graf and L. Mounier, editors, *Proc. of the 11th Int. SPIN Workshop on Model Checking of Software (SPIN'04)*, volume 2989 of *Lecture Notes in Computer Science*, pages 158–163, Spain, 2004. Springer-Verlag.
7. D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic Composition of E-services That Export Their Behavior. In M. E. Orlowska, S. Weerawarana, M. P. Papazoglou, and J. Yang, editors, *Proc. of the 1st Int. Conf. on Service-Oriented Computing (ICSOC'03)*, volume 2910 of *Lecture Notes in Computer Science*, pages 43–58, Italy, 2003. Springer-Verlag.

8. M. Bernardo, P. Ciancarini, and L. Donatiello. Detecting Architectural Mismatches in Process Algebraic Description of Software Systems. In *Proc. of the Working IEEE/IFIP Conf. on Software Architecture (WICSA'2001)*, pages 77–86, 2001.
9. J. Bézivin. MDA From Hype to Hope and Reality. Invited talk at UML'2003.
10. E. Boiten, H. Bowman, J. Derrick, and M. Steen. Viewpoint Consistency in Z and LOTOS: A Case Study. In J. S. Fitzgerald, C. B. Jones, and P. Lucas, editors, *Proc. of the 4th Int. Symposium of Formal Methods Europe (FME'97)*, volume 1313 of *Lecture Notes in Computer Science*, pages 644–664, Austria, 1997. Springer-Verlag.
11. M. Boreale, R. De Nicola, and R. Pugliese. Trace and Testing Equivalence on Asynchronous Processes. *Information and Computation*, 172(2):139–164, 2002.
12. A. Bracciali, A. Brogi, and C. Canal. A formal approach to component adaptation. *Journal of Systems and Software, Special Issue on Automated Component-Based Software Engineering*, 2004. (In press). A preliminary version of this paper was published in *Component Deployment*, LNCS 2370, pages 185–199. Springer, 2002.
13. M. Bravetti and G. Zavattaro, editors. *Proc. of the 1st International Workshop on Web Services and Formal Methods (WS-FM'04)*, Italy, 2004. To appear in ENTCS.
14. N. Busi, R. Gorrieri, and G. Zavattaro. Process Calculi for Coordination: From Linda to JavaSpaces. In T. Rus, editor, *Proc. of the Int. Conf. on International Conference on Algebraic Methodology And Software Technology (AMAST'00)*, volume 1816 of *Lecture Notes in Computer Science*, pages 198–212, USA, 2000. Springer-Verlag.
15. L. Caires. Behavioral and Spatial Observations in a Logic for the Pi-Calculus. In I. Walukiewicz, editor, *Proc. of the Int. Conf. on Foundations of Software Science and Computation Structures (FoSSaCS'04)*, volume 2987 of *Lecture Notes in Computer Science*, pages 72–89, Spain, 2004. Springer-Verlag.
16. M. Calder, S. Maharaj, and C. Shankland. A Modal Logic for Full LOTOS Based on Symbolic Transition Systems. *The Computer Journal*, 45(1):55–61, 2002.
17. C. Choppy, P. Poizat, and J.-C. Royer. Formal Specification of Mixed Components with Korrigan. In *Proc. of the Int. Conf.on Asia-Pacific Software Engineering Conference (APSEC'01)*, pages 169–176, Macau, 2001. IEEE Computer Society.
18. A. Farías and Y.-G. Gueheneuc. On the Coherence of Component Protocols. In *Proc. of the Int. Workshop on Software Composition (SC'03)*, volume 82(5) of *Electronic Notes in Theoretical Computer Science*, Poland, 2003.
19. H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based Verification of Web Service Compositions. In *Proc. of the 18th IEEE International Conference on Automated Software Engineering (ASE'03)*, pages 152–163, Canada, 2003. IEEE Computer Society Press.
20. X. Fu, T. Bultan, and J. Su. Analysis of Interacting BPEL Web Services. In *Proc. of the 13th International World Wide Web Conference (WWW'04)*, USA, 2004. ACM Press.
21. H. Garavel and R. Mateescu. SEQ.OPEN: a Tool for Efficient Trace-Based Verification. In S. Graf and L. Mounier, editors, *Proc. of the 11th Int. SPIN Workshop on Model Checking of Software (SPIN'04)*, volume 2989 of *Lecture Notes in Computer Science*, pages 158–163, Spain, 2004. Springer-Verlag.
22. D. Garlan, R. T. Monroe, and D. Wile. *Foundations of Component-Based Systems*, chapter Acme: Architectural Description of Component-Based Systems, pages 47–68. Cambridge University Press, 2000.
23. D. Garlan and Z. Wang. Acme-Based Software Architecture Interchange. In P. Ciancarini and A. L. Wolf, editors, *Proc. of the 3rd Int. Conf on Coordination Languages and Models (COORDINATION'99)*, volume 1594 of *Lecture Notes in Computer Science*, pages 340–354, The Netherlands, 1999. Springer-Verlag.

24. D. Gelernter and N. Carriero. Coordination Languages and their Significance. *Communications of the ACM*, 35(2):97–107, 1992.
25. Garavel H, F. Lang, and R. Mateescu. An Overview of CADP 2001. *EASST Newsletter*, 4:13–24.
26. R. Hamadi and B. Benatallah. A Petri Net-based Model for Web Service Composition. In K.-D. Schewe and X. Zhou, editors, *Proceedings of the 14th Australasian Database Conference (ADC'03)*, volume 17 of *CRPIT*, Australia, 2003. Australian Computer Society.
27. R. Hull, M. Benedikt, V. Christophides, and J. Su. E-Services: a Look Behind the Curtain. In ACM, editor, *Proceedings of the Twenty-Second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'03)*, pages 1–14, USA, 2003. ACM Press.
28. A. Ingolfsdottir and H. Lin. *Handbook of Process Algebra*, chapter A Symbolic Approach to Value-passing Processes. North-Holland Elsevier, 2001.
29. A. Lazovik, M. Aiello, and M. P. Papazoglou. Planning and Monitoring the Execution of Web Service Requests. In M. E. Orlowska, S. Weerawarana, M. P. Papazoglou, and J. Yang, editors, *Proc. of the 1st Int. Conf. on Service-Oriented Computing (ICSOC'03)*, volume 2910 of *Lecture Notes in Computer Science*, pages 335–350, Italy, 2003. Springer-Verlag.
30. F. Leymann. Managing Business Processes via Workflow Technology. Tutorial at the 27th International Conference on Very Large Data Bases (VLDB'01), Italy, 2001.
31. J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. John Wiley and Sons, 1999.
32. S. A. McIlraith and T. C. Son. Adapting Golog for Composition of Semantic Web Services. In D. Fensel, F. Giunchiglia, D. McGuinness, and M.-A. Williams, editors, *Proc. of the 8th International Conference on Principles and Knowledge Representation and Reasoning (KR'02)*, pages 482–496, France, 2002. Morgan Kaufmann Publishers.
33. B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid. Composing Web services on the Semantic Web. *The VLDB Journal*, 12(4):333–351, 2003.
34. N. Medvidovic and R. N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.
35. S. Nakajima. Behavioural Analysis of Component Framework with Multi-Valued Transition System. In *Proc. of the Int. Conf.on Asia-Pacific Software Engineering Conference (APSEC'02)*, pages 217–226, Australia, 2002. IEEE Computer Society.
36. S. Nakajima. Model-checking Verification for Reliable Web Service. In *Proc. of the Workshop on Object-Oriented Web Services (OOWS'02), satellite event of OOPSLA'02*, USA, 2002.
37. S. Narayanan and S. McIlraith. Analysis and Simulation of Web Services. *Computer Networks*, 42(5):675–693, 2003.
38. O. Nierstrasz. *Object-Oriented Software Composition*, chapter Regular Types for Active Objects, pages 99–122. Prentice-Hall, 1995.
39. G. A. Papadopoulos and F. Arbab. *The Engineering of Large Systems*, volume 46 of *Advances in Computers*, chapter Coordination Models and Languages. Academic Press, 1998.
40. F. Plasil and S. Visnovsky. Behavior Protocols for Software Components. *IEEE Transactions on Software Engineering*, 28(12):1056–1076, 2002.

41. P. Poizat. *Korrigan: a Formalism and a Method for the Structured Formal Specification of Mixed Systems*. PhD thesis, Institut de Recherche en Informatique de Nantes, Université de Nantes, 2000. Available at `http://www.lami.univ-evry.fr/~poizat/documents/these.ps.gz`, in French.
42. P. Poizat and J.-C. Royer. Korrigan: a Formal ADL with Full Data Types and a Temporal Glue. Technical report, Laboratoire de Méthodes Informatiques, 2003. Submitted.
43. M. Radestock and S. Eisenbach. What Do You get From a Pi-Calculus Semantics? In *Proc. of the 6th Int. Conf. on Parallel Architectures and Languages Europe (PARLE'94)*, volume 817 of *Lecture Notes in Computer Science*, pages 635–647, Greece, 1994. Springer-Verlag.
44. G. Salaün, L. Bordeaux, and M. Schaerf. Describing and Reasoning on Web Services using Process Algebra. In *Proc. of the IEEE International Conference on Web Services (ICWS'04)*, San Diego, USA, 2004. IEEE Computer Society Press. To appear.
45. G. Salaün, A. Ferrara, and A. Chirichiello. Negotiation among Web Services using LOTOS/CADP. Technical Report 13-04, DIS, Università di Roma "La Sapienza", Italy, 2004. Submitted.
46. G. Salaün and P. Poizat. Interacting Extended State Diagrams. In *Proc. of the 2nd Workshop on Semantic Foundations of Engineering Design Languages (SFEDL'04) – ETAPS'04*, Spain, 2004. To appear in ENTCS.
47. B. Schmerl and D. Garlan. AcmeStudio: Supporting Style-Centered Architecture Development. In *Proc of the 26th Int. Conf. on Software Engineering (ICSE'04)*, Scotland, 2004. IEEE Computer Society.
48. J.-L. Sourouille. Héritage et substituabilité de comportement. In *Actes de la conférence sur les Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'01)*, Nancy, France, 2001.
49. J. P. Sousa and D. Garlan. Formal Modeling of the Enterprise JavaBeans Component Integration Framework. *Information and Software Technology*, 43(3), 2001. Special issue on Component-Based Development.
50. B. Spitznagel and D. Garlan. A Compositional Formalization of Connector Wrappers. In *Proc of the 25th Int. Conf. on Software Engineering (ICSE'03)*, USA, 2003. IEEE Computer Society.
51. R. J. van Glabbeek. *The linear time - branching time spectrum I*, chapter 1, pages 3–99. Handbook of Process Algebra. Elsevier, 2001.
52. D. M. Yellin and R. E. Strom. Protocol Specifications and Component Adaptors. *ACM Trans. on Programming Languages and Systems*, 19(2):292–333, 1997.