Service Oriented Computing: a new challenge for Process Algebras

Mario Bravetti and Gianluigi Zavattaro

Dipartimento di Scienze dell'Informazione, Università di Bologna,
Mura Anteo Zamboni 7, I-40127 Bologna, Italy.
E-mail: {bravetti, zavattar}@cs.unibo.it

Abstract

Service Oriented Computing is emerging as a reference model for a new class of distributed computing technologies such as Web Services and the Grid. We discuss three main aspects of Service Oriented Computing (loose coupling, communication latency, and open endedness), and we relate them with traditional process algebra operators. We also indicate some new issues, raising from the combination of these three aspects, that require the investigation of suitable new process algebra operators.

1 Introduction

Service Oriented Computing is an emerging paradigm for distributed computing based on services as the basic computational entities. Services are autonomous, platform-independent, heterogeneous elements that interact via basic patterns of service invocation. The main novelty of service oriented computing, with respect to traditional distributed computing models, is that services are stateless and all information they need is usually passed within the exchanged messages. This technique is called contextualization because the messages contain additional context information, such as cookies or session identifiers, used to describe the state of the overall computation. Due to the statelessness assumption, the service oriented paradigm is particularly suited to program systems based on a minimal shared knowledge and understanding among the interacting parts. These systems are usually referred to as loosely coupled systems.

The most prominent service oriented technologies are Web Services and the Grid. These technologies are based on standardized mechanisms used to describe the interface of the services, to advertise and locate new services, and to invoke the available services via one of the basic interaction pattern. Complex service interactions, which cannot be trivially encoded in the basic patterns, require a so-called service orchestration. Service orchestration is usually achieved adding new components (called the orchestrators) that do not actually perform computation, but simply manage the flow of invocation of the services involved in the collaboration. A

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

plethora of languages (comprising e.g. XLANG, BizTalk, WSFL, WS-BPEL) has been recently defined to specify and program orchestrators. All these languages combine workflow constructs and communication primitives. The workflow constructs are used to decribe the flow of execution of the orchestration activities, while the communication primitives correspond to the basic service interaction patterns. Most of these languages have explicitly taken inspiration from process algebras such as CSP or the π-calculus. Nevertheless, due to peculiarities of service oriented computing, some constructs and primitives differ from the traditional operators of process algebras. Three of these peculiarities are:

Loose coupling. Orchestrators have a minimal control on the orchestrated services, for instance, a service can autonomously exit the orchestration without any previous notice.

Communication latency. The transport layer responsible for the exchange of messages, among the orchestrator and the services, does not give guarantees about the reliability and timing of remote message delivery.

Open-endedness. An orchestrator can dynamically, i.e. at run time, retrieve new services to be involved in the orchestration; for instance, this could be useful to replace services that autonomously leave the orchestration.

The remainder of the paper is organized as follows: in Sections 2, 3, and 4 we focus separately on the three above aspects, while in Section 5 we conclude discussing interesting issues raised by their combination.

2 Loose coupling

In order to cope with loose coupling, orchestration languages usually provide linguistic constructs to program the so-called loosely coupled transactions. Traditional database transactions guarantees the ACID properties: Atomicity, Consistency, Isolation, and Durability. When the activities involved in a transaction are loosely coupled the ACID properties adapt badly. In particular, Isolation usually requires to lock resources. In Web Services applications, for instance, resources may belong to different companies and there is no chance for an orchestrator to lock resources of other companies. Additionally, transactions may last long periods of time, and it is not feasible to block resources so long.

The loosely coupled transactions weaken the notion of rollback: a service might decide that rollback will not cancel all the activities carried out. The cancellation of an airplane booking, for instance, may lead to the payment of a fee. Services that do not support an “absolute” mechanism of rollback, make failures extremely complicated, to be dealt with ad-hoc rollbacks. These ad-hoc rollback processes are called compensations.

The notion of compensation is the key aspect of several recent processs algebras defined on purpose to formalize the semantics of compensation execution, and to reason about properties of compensation policies. The first proposal in this direction is StAC, a calculus with an explicit compensation operator whose operational semantics has been formalized in [9]. StAC has recently inspired also a new CSP dialect, called cCSP [10], whose semantics is defined denotationally in terms of traces. An alternative proposal is represented by the SAGAS calculi [8] that defines a concurrent big-step semantics for sequential, parallel, and nested compensatable transactions. Recently, in [7] cCSP and the SAGAS calculi have been thoroughly compared discussing how to encode (fragments of) the former in (some of) the latter calculi, and vice versa. Compensations have been formalized and investigated also in the context of π-calculus in [5], where a calculus inspired by the compensation policy of BizTalk is presented.

3 Communication Latency

Orchestration languages support a time aware programming style. For instance, in the visual orchestration language BizTalk, timed activities can be programmed which are interrupted in case they do not complete within a predefined period of time. Similarly, in BPEL4WS, it is possible to program signals that are raised at specific time instant, and to install handlers that are triggered by these signals.

Timed process algebras are an extremely powerful tool for modeling and analysing timed systems. There exists numerous models of time inspired by different intuitions and abstractions, see e.g. [1] for a comprehensive overview. In [2] a model of time particularly suited for loosely coupled distributed computing has been defined and investigated in the context of π-calculus. The proposed model can be briefly summarized as follows: processes are distributed across node networks and each node has its own clock; these clocks are not synchronized; and each action occurring within a node takes one time unit. The unique time aware operator which is considered is the classical timed out input operation: if an input does not occur within a specified delay, an alternative process is activated.

4 Open-endedness

Open-endedness is an inherent characteristics in orchestration of services retrieved from the internet: new services may appear and disappear at run-time, available services (or their efficiency) may depend on their current location or on the current location of the orchestrator (if we deal with mobile entities), requests towards services offering the same service (where "same" is established in terms of some semantical definition of its behavior) may be distributed so to have a balanced workload. Assuming that we know available services and we bind them when the orchestrator is created (i.e. at "compile-time") is not realistic in this context.

Expressing open-endedness in process algebra requires evolved mechanism for channel retrieval to access services. In particular the retrieval should be based on requirements on the desired service, e.g. on some abstraction of its behavior. This can be done in several ways: by using matching rules on tuples of data (formed e.g. by one element representing the channel and others describing the service and its behavior) as in Linda [12] or by using direct subtyping on channels themselves [11].

Note that in this context process algebra may be involved even in the description of the desired behavior of services itself. For example [13] uses abstract process algebraic descriptions as types of systems (services in our case) which are expressed in a more complex process algebra.

5 Conclusion

In this section we discuss some interesting issues raised by the combination of the three above aspects. In particular, we first discuss the combination of loosely coupled transactions and time, then we consider interesting aspects related to the combination of transactions and open-endedness, and finally we discuss how timing issues and open-endedness can be combined.

Timed transactions are linguistic constructs supported by most of the orchestration languages. A timed transaction is an activity which is interrupted if not completed within a specified time-out, and substituted by an ad hoc failure handler. This construct contrasts with the typical time-out operator discussed in Section 3. Indeed, a timed out process in [2] does not execute actions during the elapsing of the time-out. On the other hand, in timed transactions, activities are executed during the elapsing of the time-out and possibly rolled back or compensated if the overall activity is not completed in due time. Timed transactions have already received a preliminary process algebraic formalization in [14], but several issues (such as suitable observational semantics for processes distributed across networks, or a time model supporting different time costs for different kinds of operations) still require investigation.

As far as transactions and open endedness are concerned, it is interesting to observe that in service oriented computing there is a great interest in negotiation and contract protocols. These are used to select the partners involved in a transaction according to some minimal service requirements. Process algebras have been already used to model specific negotiation protocols used in the context of distributed commitment. The two phase commitment protocol guaranteeing atomicity is analysed in [3], while the BTP protocol guaranteeing a relaxed notion of partial atomicity –called cohesion– was investigated in [4]. These are only specific cases of negotiation protocols; a formal investigation of other protocols such as those based on quality of services, or supporting the dynamic redefinition of the involved partners during the execution of the transaction, is still lacking.

As far as timing issues and open-endedness are concerned, probabilistic mechanisms in process algebra combined with mechanism for open-endedness (e.g. based on matching of tuples representing service instances as in Linda) can be exploited to express dynamic retrieval of services which distribute workload of requests. In particular the deliver of requests can be probabilistically distributed among the different instances of services which possess the required behavior by means of a probabilistic distribution which is based on their individual performance (see [6]). A dynamic reconfiguration of such probabilistic distribution based on an evaluation of current performance of services is, however, still lacking.

References

[1] J.C.M. Baeten and C.A. Middelburg. Process Algebra with Timing. EATCS Monograph, Springer Verlag 2002.

[2] M. Berger. Basic theory of reduction congruence for two timed asynchronous π-calculi. In *CONCUR'04, Proc. of the 15th International Conference on Concurrency Theory*, volume 3170 of *LNCS*, pages 115–130. Springer-Verlag, 2004.

[3] M. Berger and K. Honda. The Two-Phase Commit Protocol in an Extended π-Calculus. In *EXPRESS'00, Proc. of the 7th International Workshop on Expressiveness in Concurrency*, volume 39 of *ENTCS*, 2000.

[4] L. Bocchi. Compositional Nested Long Running Transactions. In *FASE'04, Proc. of the Fundamental Approaches to Software Engineering*, volume 2984 of *LNCS*, pages 194–208. 2004.

[5] L. Bocchi, C. Laneve, and G. Zavattaro. A calculus for long running transactions. In *FMOODS'03, Proc. of the 6th IFIP International Conference on Formal Methods for Open Object-based Distributed Systems*, volume 2884 of *LNCS*, pages 124–138. Springer-Verlag, 2003.

[6] M. Bravetti, R. Lucchi, G. Zavattaro, R. Gorrieri *"Web Services for E-commerce: guaranteeing security access and quality of service"*, in Proc. of the *19th ACM Symposium on Applied Computing (SAC'04), special track on E-Commerce Technologies*, H. Haddad, A. Omicini, R.L. Wainwright and L.M. Liebrock eds., ACM Press, pp. 800-806, Nicosia (Cyprus), March 2004

[7] R. Bruni, M. Butler, C. Ferreira, T. Hoare, H. Melgratti, and U. Montanari. Reconciling two approaches to compensable flow composition. In *CONCUR'05, Proc. of the 16th International Conference on Concurrency Theory*, volume to appear of *LNCS*, 2005.

[8] R. Bruni, H. Melgratti, and U. Montanari. Theoretical foundations for compensations in flow composition languages. In *POPL'05, Proc. of the 32nd Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages*, 2005

[9] M. Butler and C. Ferreira. An operational semantics for StAC, a language for modelling long-running business transactions. In *COORDINATION'04, Proc. of the 6th International Conference on Coordination Models and Languages*, volume 2949 of *LNCS*, pages 87–104. Springer-Verlag, 2004.

[10] M. Butler, T. Hoare, and C. Ferreira. A trace semantics for long-running transactions. In Proceedings of 25 Years of CSP, London, 2004.

[11] G. Castagna, R. De Nicola, and D. Varacca. Semantic subtyping for the pi-calculus. In *LICS'05, Proc. of Logic in Computer Science*, june 2005.

[12] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[13] A. Igarashi, N. Kobayashi. A Generic Type System for the Pi-Calculus. In Theoretical Computer Science, 311(1-3):121-163, 2004.

[14] C. Laneve, and G. Zavattaro. Foundations of Web Transactions. In *FOSSACS'05, Proc. of the 8th Foundations of Software Science and Computational Structures*, volume 3441 of *LNCS*, pages 282–298. Springer-Verlag, 2005.