Contract-Oriented Software Development for Internet Services

by Pablo Giambiagi, Olaf Owe, Anders P. Ravn and Gerardo Schneider

The 'COSoDIS' project - Contract-Oriented Software Development for Internet Services - is developing novel approaches to implementing and reasoning about contracts in service-oriented architectures (SOA). The rationale is that system developers benefit from abstraction mechanisms in working with these architectures. Therefore the goal is to design and test system modelling and programming language tools to empower SOA developers to deploy highly dynamic, negotiable and monitorable Internet services.

As recently as several years ago, technology gurus predicted that the next big trend in software system development would be service-oriented architecture: that is, a successful integration of loosely coupled services belonging to different organisations, sometimes competing but on specific tasks collaborating, would storm the world. This would create a myriad of new business opportunities, enabling the formation of virtual organisations in which small and medium-sized enterprises would join forces to thrive in an increasingly competitive global market. The dream lives on, and the industry is pouring resources into developing and deploying Web services. Yet the degree of integration achieved between different organisations remains low. Collaboration presumes mutual trust, and wherever trust is not considered sufficient, business people turn to contracts as a mechanism to reduce risk. In other words, for the SOA to deliver its promised advantages, developers need cost-effective contract management solutions.

The main problems and open issues identified for supporting Web services development are the following:

1. Formal definition of generic contracts: currently, no unified system of syntax and semantics exists for contracts (in particular for quality of service (QoS) and security).
2. Negotiable and monitorable contracts: a contract must be negotiated until both parties agree on its final form, and it must be monitorable in the sense that there must be a way to detect violations. Current programming languages provide little support for negotiable and monitorable contracts.
3. Combination of object-orientation and concurrency models based on asynchronous message-passing. The shared-state concurrency model is not suitable for Web service development.
4. Integration of XML into a host language, reducing the distance between XML and object data models.
5. Harmonious coexistence at the language level of real-time and inheritance mechanisms.
6. Verification of contract properties: the integration of contracts in a programming language should be accompanied by the ability to guarantee essential properties. Guaranteeing the non-violation of contracts can be done in (at least) four different ways: (a) with runtime enforcement (eg through monitors); (b) by construction (eg through low-level language mechanisms); (c) with static program analysis techniques; or (d) through model checking.

None of the above can be used as a universal tool; they must be combined.

In addressing these issues and problems, we need to develop a model of contracts in an SOA that is broad enough to cater for at least QoS and security contracts. A basic requirement is the ability to seamlessly combine real-time models (for QoS specifications) and behavioural models (essential to constrain protocol implementation and enforce security). Contract models should also address discovery and negotiation. However, the formal definition of contracts should only be a first step towards the more ambitious task of providing language-based support for the programming and effective usage of such contracts.

Some contracts may be seen as wrappers that 'envelop' the code/object under the scope of the contract. Firewalls, for instance, may be seen as a kind of monitor of the contract between a machine

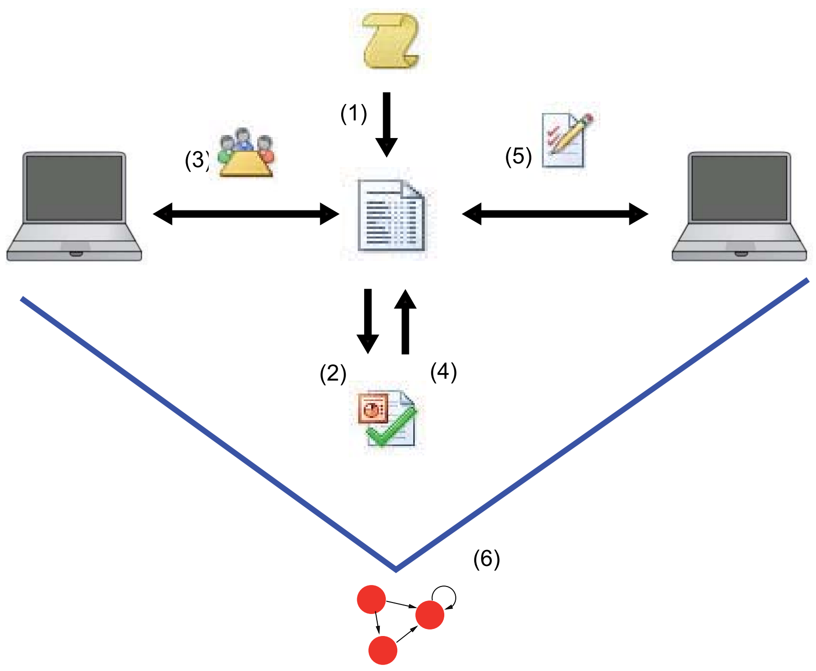

Figure 1: A contract (template) is generated in an electronic version (1); the contract is checked to be free of contradictions (2); a negotiation starts (3); different versions of the contract are checked (4); a final contract is signed (5); a runtime monitor guarantees the contract fulfilment (6) .

and the external applications wanting to run on that machine. It would be interesting to investigate a language primitive to create wrapped objects that are correct by construction.

Contracts for QoS and security could also be modelled as first-class entities using a 'behavioural' approach, through interfaces. In order to tackle time constraints (related to QoS), such interfaces need also to incorporate time.

Finding languages or notations for describing timing behaviour and requirements is easy: the real challenges are in analysis. Besides syntactic extensions, the language needs to have time semantic extensions in order to allow extraction of a timed model, eg a timed automaton. This model may be checked with existing tools such as Kronos or Uppaal, while other properties may instead be proven correct by construction (eg wrappers).

In practice, many properties can only be validated through runtime approaches. A promising direction is to develop techniques for constructing a runtime monitor from a contract, which will be used to enforce its non-violation (cf ongoing work with Java Modeling Language and Spec#).

In summary, our aim is to develop language-based solutions to address the above problems through the formalization of contracts as enriched behavioural interfaces, and the design of appropriate abstraction mechanisms that guide the developer in the production of contract-aware applications.

The COSoDIS project is a Nordic project funded by the Nordunet3 committee. The partners involved are the University of Oslo (Norway), Aalborg University (Denmark) and SICS (Sweden).

Link:
http://www.ifi.uio.no/cosodis/

Please contact:
Gerardo Schneider
University of Oslo, Norway
Tel: +47 22 85 29 71
E-mail: gerardo@ifi.uio.no
http://folk.uio.no/gerardo/

gCube: A Service-Oriented Application Framework on the Grid

by Leonardo Candela, Donatella Castelli and Pasquale Pagano

gCube is a new service-oriented application framework that supports the on-demand sharing of resources for computation, content and application services. gCube enables the realization of e-infrastructures that support the notion of Virtual Research Environments (VREs), ie collaborative digital environments through which scientists, addressing common research challenges, exchange information and produce new knowledge. gCube is currently used to govern the e-infrastructure set up by the European integrated project DILIGENT (A Digital Library Infrastructure on Grid-Enabled Technology).

The long-term journey towards the e-science vision demands e-infrastructures that allow scientists to collaborate on common research challenges. Such infrastructures provide seamless access to necessary resources regardless of their physical location. These shared resources can be of very different natures and vary across application domains. Usually they include content resources, application services that manipulate these content resources to produce new knowledge, and computational resources, which physically store the content and support the processing of the services. Many e-infrastructures already exist, at different levels of maturity, and support the sharing of and transparent access to resources of a single type, eg SURFNet (information), GriPhyN (services), EGEE (computing and storage). However, they are still too primitive to support a feasible realization of VREs. The DILIGENT infrastructure, released recently by the homonymous project, will overcome this limitation by supporting in a single common framework the sharing of all three types of resource.

The core technology supporting such e-infrastructure is a service-oriented application framework named gCube. gCube enables scientists to declaratively and dynamically build transient VREs by aggregating and deploying on-demand content resources, application services and computing resources. It also monitors the shared resources during the lifetime of the VRE, guaranteeing their optimal allocation and exploitation. Finally, it provides mechanisms to easily create dedicated Web portals through which scientists can access their content and services.

From the technological point of view, gCube provides: (i) runtime and design frameworks for the development of services that can be outsourced to a Grid-enabled infrastructure; (ii) a service-oriented Grid middleware for exploiting the Grid and hosting Web Services on it; (iii) a set of application services for distributed information management and retrieval of structured and unstructured data.

Runtime frameworks are distinguished workflows that are partially pre-defined within the system; they include Grid-enabled services and application services, where the former coordinate in a pure distributed way the action of the latter, while relying on a high-level characterization of their semantics. Design frameworks consist of patterned blueprints, software libraries and partial implementations of state-of-the-art application functionality, which can be configured, extended and instantiated into bespoke application Grid services.

The service-oriented Grid middleware provides all the required capabilities necessary to manage Grid infrastructures. It eliminates manual service deployment overheads, guarantees opti-