

An Organisational Approach to Building Adaptive Service-oriented Systems

Alan Colman and Jun Han

Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne, Victoria, Australia
{acolman,jhan}@swin.edu.au

Abstract. The relationships between the loosely coupled services of a composition can be non-deterministic and unreliable. This requires that a composite application have the ability to adaptively reconfigure itself to continuously satisfy the system requirements. Imperative composition approaches, such as BPEL, do not provide the abstractions necessary to create adaptive compositions. In this paper we introduce an organisation-oriented, application-centric service composition framework that aims to achieve adaptive application behaviour from non-deterministic and unreliable services. The responsibility for coordinating and managing service interactions resides with a coordination/management layer of the application itself. This layer involves two types of essential elements: service contracts and service organisers. The service contracts define and regulate the relationships between the services in a composition. The service organisers control the bindings between service types and individual services, and can create and revoke contracts between services within the composition in order to adaptively reconfigure the application in response to changing conditions. An adaptive service composition can be formed by connecting and configuring services through dynamically formulated contracts, under the control and management of service organisers.

1 Introduction

In order to compose Web services with other services or applications, the services need to be functionally compatible, the interactions between the services need to be well-ordered, and the composed behaviour must meet any requirements for the system as a whole. However, in the open and distributed environment of the Internet, the relationship between loosely coupled services can be non-deterministic. In an application-centred architecture, the central application needs to be able to meet its functional and non-functional requirements, even if the Web services it relies upon are to some degree unpredictable. In particular, if an application needs to guarantee a certain level of performance, and yet relies on distributed Web services to provide some functionality, the application needs a mechanism to manage the quality of service (QoS) of its subsidiary services. For example, if the application is a work flow

scheduling system, the system needs to respond dynamically to variations in the performance of its constituent services. It must monitor and restructure the interactions of the services as demand changes and service loads change. As such, it requires the composite system to have *management* capabilities.

Much work has been done on the preparation of standards to allow the development of Web service compositions that are well-behaved and meet the requirements of the composite system. While the point-to-point simple interactions have reached a level of maturity in their standardisation and implementation, achieving reliable behaviour from more complex configurations of Web services is still an open problem. The development of standards to address such behavioural and non-functional aspects of complex composed services is being undertaken on a number of fronts: for example, coordination [3], choreography [11], orchestration [4], and management [10].

Many of these standardisation efforts define a management or coordination level of abstraction. In these standards management and coordination functions are encapsulated as services with a well-defined Web service interface. No clear guidance is provided on how the programmer can use these various overlapping and sometimes incompatible middleware standards to achieve a well regulated, adaptive application. What is needed is a framework that provides the necessary management and coordination functions that the application programmer can readily extend to suit domain and application specific requirements. In this paper, we propose such a framework – the Role Oriented Adaptive Design (ROAD) framework for Web services. Like a number of WS standards, the ROAD framework defines a separate coordination/management layer. However, this layer is superimposed on the pre-existing decoupled application entities and creates an organisational structure for it, rather than being a separate encapsulated service provided by the middleware. The primary responsibility for coordination and management of the interactions resides with this coordination/management layer of the application.

The structure of the paper is as follows. Section 2 defines the concept of an adaptive system in terms of managed indirection of *instantiation* and *composition*. Section 3 outlines a role-based architectural framework that supports managed indirection to achieve adaptivity. Section 4 concludes and briefly discusses of related work.

2 Adaptive systems – the management of indirection

We define an adaptive system as one that can reconfigure itself in response to changes in its environment, or in response to changes in its own goals or capabilities. The perturbation between a system and its environment can include the availability and variable performance of Web services that it relies on to function. Even if a system's services have unpredictable performance, we still need to achieve some acceptable level of system-wide performance. A system needs to be able to recognise when one of its services is underperforming, and then take remedial action: action such as reorganising the work load, reconfiguring the interactions between services, or replacing the underperforming service. Adaptable systems are those that can be

readily recomposed. To achieve recomposition, the system structure must be flexible, with indirection between the entities in the system.

Indirection in Web service compositions can be created in two dimensions — *indirection of instantiation*, and *indirection of composition*. Firstly, abstract description of services can be distinguished from their concrete implementation (as in the separation of the abstract and concrete parts of a WSDL specification [12]). In an adaptive system, such binding should be able to be created and destroyed at runtime. Such dynamic binding is well supported in Web services (in specification at least) through service discovery and selection mechanisms.

The second type of indirection is found in the association between the abstract services. Associations between service types might be hard-coded references in the implementation of a client service, or hard-coded using an imperative composition language such as BPEL [4]. Such associations are abstracted by using reference variables, or by explicitly representing an association type (such as a contract or partner-link type as in BPEL). However, in the absence of other mechanisms, all possible runtime *determinations* (resolutions) of the association indirection must be anticipated at design time (e.g. in BPEL as cases in a switch structure). The indirection in the association is not *per se* adaptive because the associations are hard-coded when the service or the composition script is implemented.

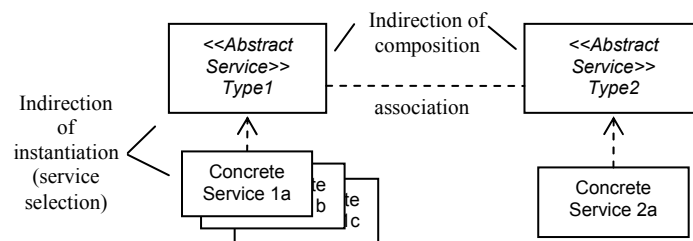


Fig. 1 Two dimensions of indirection in Web services.

Fig. 1 illustrates the two types of indirection that give software systems the flexibility needed to be adaptive: namely indirection of *instantiation* and *composition*. While the mechanism of runtime determination of instantiation is supported in Web services, the determination of composition between service types is often fixed during implementation.

In a functioning system any indirection must be determined before or during runtime. An adaptive system needs to manage its indirection: that is, it needs to be able to dynamically create bindings between its loosely coupled elements on both the above dimensions in response to changing demands and changing environment. To do this, the adaptive system needs to be able to perform the following management and binding functions:

- monitor the performance (or other quality attributes) of its current configuration;
- have access to alternative services (with various performance/quality characteristics) and the ability to select between services
- ensure associated services are functionally and interactionally compatible
- check the validity and evaluate the performance of alternative compositions
- reconfigure its bindings and associations.

The basic Web service standards of WSDL and UDDI partially address some of these functions (basic functional compatibility and service selection respectively). There has been a recent proliferation of standards that address some of the other functions listed above [3,8,10,11]. These standards are to an extent overlapping and incompatible, and they do not provide clear guidance on how to integrate and use such standards, if we want to develop an adaptive application as characterised above. For example, BPEL allows us to define compositions from services that are compatible in signature, but does not support performance monitoring in its orchestration model other than fault handling and compensation. If the programmer wants to monitor the performance of a service, the monitoring code is tangled with the orchestration script. As well as providing the necessary indirection, an adaptive architecture must show what elements in the system are responsible for the binding and management of that indirection, and how this can achieve adaptive adjustment.

3 An Adaptive Framework for using Web Services

In this section we introduce a framework to facilitate the development of applications built from Web services. We call this framework Role Oriented Adaptive Design (ROAD). The design aims of this framework are as follows:

- facilitate the development of adaptive applications that can perform reliably while using Web services whose performance is unreliable
- support the managed indirection of instantiation and composition
- provide a framework of constructs that can be extended by the programmer to create coordination and management for adaptive composites
- allow the management/coordination layer to be coded as a separate concern from the functional roles and services
- have compositions that can be imposed on services without the services needing to be (necessarily) written to horizontal standards such as WS-Coordination.

This section outlines the basic elements of the ROAD framework, and shows how it can be used to create adaptive compositions as defined above. The adaptive framework for using Web services introduced here is based on the conceptual separation of *roles* from the entities that play those roles, and the association of such roles with contracts. Applications are viewed as organisations — *goal-driven* networks of roles bound together by contracts. Roles can be played by various *players*, in much the same way as a role in a business structure may be played by various employees, or outsourced to external organisations. Similarly, roles in the adaptive service-oriented application can be played by Web services or other components that are within the organisation, or by external Web services outside the application's immediate scope of control. Players can be dynamically bound/unbound (indirection of instantiation) to roles as demands on the application changes, or as the players' performance varies.

Contracts associate organisational roles. They monitor and regulate interactions between the roles. As all roles (as opposed to the players) are internal to the organisation, contracts are also internal (although these internal contracts may be mirrored in external SLAs). Contracts define the mutual obligations of the participant roles in an organisational context. They define what interactions are permissible or

required by the participant roles, and can be used to enforce sequences of interactions (conversations). Contracts can also be used to set performance or other quality conditions on the roles' interactions, and monitor those interactions for compliance to those conditions (performance management). Contracts thus encapsulate both the coordination and the performance management of interactions.

Organisers make and break the bindings between organisational roles and players (service selection), and create and revoke the contracts between the roles. They can thereby create various configurations of roles and players. Organisers set performance requirements for the contracts they control, and receive performance information from those contracts. Organisers are themselves a role-player pair, so that role players of varying capability can be bound to the organiser role. In short, organisers provide the adaptivity to the application by managing the indirection of composition and instantiation. Organisers are responsible for the configuration of a set of roles and contracts. We call such configurations *self-managed composites*. The following subsections explain these concepts in more detail.

3.1 Determining Indirection with Bindings and Contracts

At runtime, services are bound to roles, and roles are associated with each other using contracts. Functional roles are application entities that hold abstract service definitions. Because the services that play roles can be transitory (for example, there may be no service currently available to play a role), roles also maintain any state (e.g. message queues) associated with their position in the organisation. To illustrate the concept of a role, consider a highly simplified segment of a production system application for a Widget manufacturing department. Thingies are obtained from an external supplier and made into Widgets by an Assembler. A Foreman supervises, among other things, the work schedule of the Assembler.

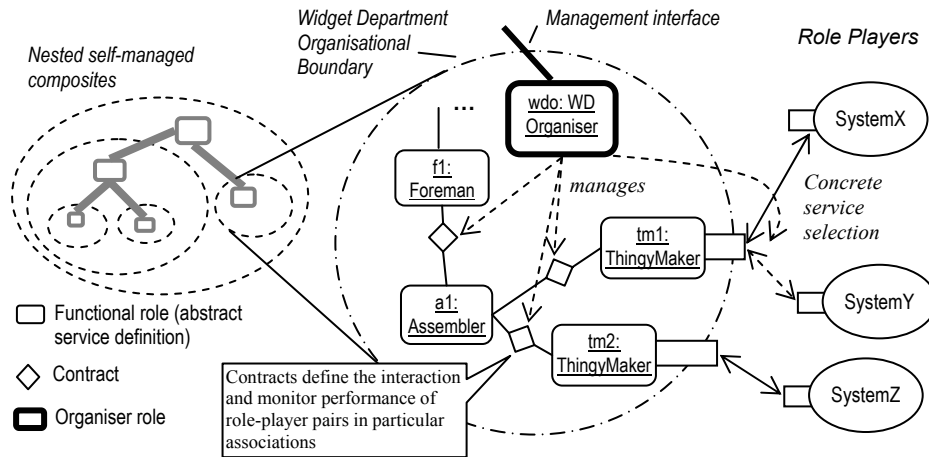


Fig. 2. Creating flexible role-structures with role-player bindings and role-role contracts

Fig. 2 above illustrates the relationship between a role within an organisation and an external service that plays that role. The role can be thought of as a proxy within the

organisation for the external service. The role includes a WDSL abstract service definition that is made concrete with a service endpoint reference when a service is bound to the role. Service selection (eg. between ServiceX and ServiceY) is not the responsibility of the role itself, but the responsibility of the composite's organiser wdo (as discussed below).

Contracts between roles can also be used to create flexibility. By creating and/or revoking contracts, the topology of the composition of roles can be altered. For example, if a role instance tm1 of type ThingyMaker (played by ServiceX) is unable to meet the requirements of an Assembler a1. An alternative to replacing ServiceX with a more capable service (say ServiceY) would be to add another role instance tm2 of type ThingyMaker (played by ServiceZ) to the role structure. This changes the topology of the organisation. Contracts, however, provide more than simple references between role types. They monitor and regulate interactions between the roles. Contracts define the mutual obligations of the participant roles. They define what interactions are permissible or required by the participant roles, and can be used to enforce sequences of interactions. Contracts can also be used to set performance and quality conditions on the interactions, and monitor those interactions for compliance to those conditions (performance management). Contracts thus encapsulate both coordination and performance management of interactions. A more detailed discussion of ROAD contracts, their contents and how they are implemented can be found in [6,7].

Functional contracts specify the required performance of interactions between the parties, and they monitor and store actual performance. They can be thought of as service level agreements (SLAs) for the roles in the organisation. The monitoring of service performance using the ROAD framework is extrinsic; that is, it is performance of a concrete role-player pair *as measured from* the application that is using the service. The application does not have to rely on abstract descriptions of service quality (although this may be helpful as a starting point for service discovery) because it can measure the time elapsed (or other change of state) between, for example, request and response messages. Basic time and count based metrics (similar to those defined in WSDM-MoWS [10]) are provided as part of the ROAD framework and do not have to be coded by the application programmer. However, the programmer can extend the performance measurement capabilities with domain-specific metrics (for example cost or reliability functions) if required. There is another advantage to storing performance measurements in the contract rather than in, say, a log of *all* interactions of a service. The advantage of contract-centric monitoring is that the performance profile of a service relates to a particular client. This profile may vary depending on the client identity and location of the service; for example the client's priority or the capacity of the connection between the client and the service.

3.2 Internal Contracts and External SLAs.

The above discussion only deals with 'internal' contracts between roles within the organisational boundary, although those roles may be proxies that represent external services. Some interactions, particular long-lived ones, may need an external contract between the role (Web service proxy) and player (concrete Web service). The ROAD framework does not preclude the use of external coordination mechanisms such as WS-Coordination [3], or external contracts such as WS-Agreement [8]. Indeed, the

internal contract that binds the proxy role to the rest of the system can serve as a template for defining external coordination contexts or contracts.

The concept of a ROAD contract also maps naturally to Service Level Agreements at the business level, in part because roles (such as ThingyMaker) map better to business entities than do, say, BPEL activities. Discussion of these mappings is beyond the scope of this paper, other than to note that the definition of an internal contract provides the desired requirements (both protocol and performance) from the organisation's point of view. What can be negotiated in an external contract also provides constraints for what can be defined in the associated internal contract.

3.3 Organisers and Self-managed Composites

A self-managed composite contains functional roles bound by contracts and an *organiser* role. From the enclosing system's perspective, a self-managed composite is a role player with a management interface. Self-managed composites contain roles which may themselves be self-managed composites. This recursive structure enables the nesting of self-managed composites (as in Fig. 2). Self-managed composites have two interfaces – a *functional* interface and a *management* interface. The *functional* interface of a composite is the aggregation of all the Web services it exposes. The *management* interface of the composite is the interface of the *organiser* role. This is a conceptually similar arrangement to an 'out-of-band' manageability interface in MoWS [10] which could be used for this purpose.

The organiser role is responsible for monitoring the performance of the composite's contracts either through notification from its contracts or by polling them. It also reconfigures the composite if environmental perturbation, or a change of requirements, leads to the composite not meeting (or potentially not meeting) its contractual performance obligations. The organiser can achieve reconfiguration by assigning and revoking contracts, and by changing the binding between functional roles and services as shown in Fig. 2. For an organiser to be able to respond adaptively to changing situations, it needs to have at its disposal a range of role/role-players of various performance characteristics, and a mechanism for service discovery such as UDDI. The function of organisers and the dynamics of reconfiguration is not further described here due to space limitations, but a description of adaptive behaviour in self-managed composites, can be found in [5].

4 Related Work and Conclusion

The ROAD adaptive framework for using Web services can be used by an application programmer to create applications that respond to changes in performance or other non-functional requirements. The application can also adapt to performance variations in the Web services it uses. In ROAD, contracts are used to monitor performance/reliability, and enforce protocols. Organisers provide adaptive behaviour to the application by creating and revoking contracts between roles, and by binding roles to services. Organisers control a cluster of roles, contracts and service bindings called a self-managed composite. These composites form a recursive hierarchy that localises organisation but allows system level requirements to be fulfilled.

Like BPEL, our framework is a means of creating executable compositions, but ROAD has roles as its fundamental unit rather than activities. The use of contracts in our framework maps more naturally to inter-organisational business contracts than do process-based approaches. While BPEL is not in itself adaptive, there has been much recent focus on making service composition more flexible. A recent overview of dynamic workflow-based composition can be found in [13]. These approaches focus on adaptive processes using process abstraction, rather than an adaptive role structure as presented here. Monitoring of services has also been addressed in [2] using external ‘Smart Monitors’ services rather than application-based monitors. However, there is no mechanism for ‘monitoring the monitors’ (organisers and contracts) as in the recursive ROAD structure. The Cremona framework [9] based on WS-Agreement addresses many of the same issues as the ROAD framework, but focuses on external contracts. These contracts do not control interaction as the management level of a ROAD contract does.

While we have not discussed the implementation of the ROAD framework in this paper, [7] gives a description of an aspect-oriented implementation of ROAD contracts. A similar aspect-oriented approach can be found in [1], where policies are woven into the Web service stubs rather than used to create instances of contracts.

References

- [1] Baligand, F. and Montfort, V., "A concrete solution for web services adaptability using policies and aspects," *Proc. of the 2nd Inter. Conf. on Service Oriented Computing*, 2004.
- [2] Baresi, L., Ghezzi, C., and Guinea, S., "Smart Monitors for Composed Services," *Proc. of the 2nd Inter. Conf. on Service Oriented Comp (ICSOC'04)*, 2004.
- [3] BEA Systems, IBM, and Microsoft *Web Services Coordination (WS-Coordination)*, <http://msdn.microsoft.com/library/en-us/dnglobspec/html/WS-Coordination.pdf> (2004)
- [4] BEA, IBM, Microsoft, SAP, AG, and Siebel Systems *Business Process Execution Language for Web Services (BEPL4WS)*, <http://ifr.sap.com/bpel4ws/index.html> (2003)
- [5] Colman, A. and Han, J., "Coordination Systems in Role-based Adaptive Software," *Proc. of the 7th Inter. Conf. on Coordination Models and Languages*, LNCS 3454, 2005.
- [6] Colman, A. and Han, J., "Operational management contracts for adaptive software organisation," *Proc. of the Australian Software Eng. Conf. (ASWEC 2005)*, 2005.
- [7] Colman, A. and Han, J., "Using Associations Aspects to Implement Organisational Contracts," *Proc. of the 1st Inter. Workshop on Coordination and Organisation*, 2005.
- [8] Global Grid Forum *Web Services Agreement Specification (WS-Agreement)*, www.gridforum.org/Meetings/GGF11/Documents/draft-ggf-graap-agreement.pdf (2004)
- [9] Ludwig, H., Dan, A., and Kearney, R., "Cremona: an architecture and library for creation and monitoring of WS-agreements," *Proc. of the 2nd Inter. Conf. on Service Oriented Computing (ICSOC'04)*, 2004.
- [10] OASIS *Web Services Distributed Management -Management of Web Services 1.0*, wsdm-mows-1.0, <http://docs.oasis-open.org/wsdm/2004/12/cd-wsdm-mows-1.0.pdf> (2005)
- [11] W3C *Web Services Choreography Description Language (WS-CDL) WD-ws-cdl-10-20041012*, <http://www.w3.org/TR/ws-cdl-10/> (2004)
- [12] W3C *Web Services Description Language (WSDL) Version 2.0 Primer*, WD-wsdl20-primer-20050510, <http://www.w3.org/TR/wsdl20-primer/> (2005)
- [13] Zirpins, C., Lamersdorf, W., and Baier, T., "Flexible coordination of service interaction patterns," *Proc. of the 2nd Inter. Conf. on Service Oriented Comp (ICSOC'04)*, 2004.