

A Multi Faceted Management Interface for Web Services

Justin King and Alan Colman

*Centre for Complex Software Systems & Services
Swinburne University of Technology
Melbourne, Victoria, Australia
{jinking, acolman}@swin.edu.au*

Abstract

In open systems, independent services exist in administrative domains outside that of the consumers of those services. Current standards exist to create management interfaces to allow operational management of services while keeping the services basic functionality fixed. These standards lack the ability to facilitate the establishment of agreements between services across administrative domains or with a divergent level of autonomy and adaptability. In this paper we propose a multi-faceted management interface capable of both operational and contractual operations. This allows relationships to be established between services with varying levels of capability. We then apply this management interface to an existing architectural framework that explicitly represents agreements as contracts between services and manages the interferences between those contracts.

1. Introduction

Service-oriented architectures (SOAs) are based on the concept of loosely coupled compositions of well-described services that are bound together according to policies and managed at runtime. These loosely coupled services often are ‘*independent*’ in the sense they have different controllers or owners. Implementations of applications such as supply chain systems often compose services that cross administrative and organisational domains. This multi-administrative/multi-ownership nature of compositions raises problems for conventional approaches to software and device management. There is a need for frameworks that support service coordination across domains both at the management level as well as the functional level. In open systems, these frameworks will need to support common well-defined management communication protocols. These protocols need to address the *establishment* and *maintenance* of management

relationships. Within the context of an established management relationship, these protocols also need to support *operational* management communication that maintains functional relationships.

There are many types of management relationships that can exist between services of various granularities in an SOA composition. At a fine level of granularity, services can be used to wrap the functionality of basic resources such as data sources while at higher levels of granularity services may be complex self-managed systems or wrap legacy applications with extensive configuration options. A generalised management framework will need to be able to support the relationships between independent services with various levels of autonomy and self-management.

In this paper we focus on the management of *independent* services that have various levels of *autonomy* and *adaptivity*. Such services may range from basic services that present no management options; to configurable services that provide a set range of configurable parameters or modes; to services that are self-managed and attempt to adapt to the demands of their environments. When composed together, these various types of services will have between them various types of management relationships that express various levels of control and autonomy between the services. This heterogeneity of such management relationships means that the *facets* of management relationships need to be explicitly defined. We will define a conceptual schema that categorises these facets, show how a management interface that incorporates these facets can be defined by extending the WS-Management standard [1], and describe how a management interface that can incorporate such facets can be incorporated into a framework that supports service compositions.

The paper is structured as follows. Section 2 briefly examines some existing approaches to management communication and where they lack the ability to function in situations across multiple administrative domains. Section 3 presents a motivating scenario where management communication supports driving tasks across independent services with different owners.

In Section 4 we define the levels of management capability and associated message types a service may require. Section 5 proposes a standard management interface and an implementation strategy for a management framework that supports management contracts between independent services in a composite. This utilises the contract-based composites provided by the ROAD framework [2]. We conclude with Section 6.

2. Management standards and approaches

A number of approaches and frameworks have been developed in recent years to facilitate the management of distributed systems. For example, IETF's SNMP (Simple Network Management Protocol) [3] is a TCP/IP based management protocol for network devices such as routers, switches and servers. SNMP is considered the de facto standard for network management [4]. Management related information on these resources is kept in a MIB (Management Information Base), which is accessed by management agents which expose the MIB to SNMP clients. Clients may then query or update these values. They may also subscribe to notifications for changes in those values. In this sense SNMP is an *operational* management standard as defined above. Similarly, the Common Information Model (CIM) [5] extends SNMP by being able to define schemas for managed applications and systems as well as devices. It also enables the definition of associations between managed components. WBEM (Web-Based Enterprise Management) [6] works with CIM to provide a standard set of Web based operations and tools that can be used to manipulate managed entities. All these management standards and protocols have focused on remotely managing devices and systems that exist under the same administrative domain as the manager of that resource. Other shortcomings in relation to management of SOAs is that they are too low level to allow flexible, coordinated interaction patterns involving business processes [4].

Another recent approach to the management of software systems has been to envisage and design systems as self-managed or autonomic [7]. Autonomic systems have some internal management capability responsible for the smooth operation of the resource, rather than being remotely managed as with SNMP and CIM. However, in applying the precepts of autonomic computing to the domain of service-oriented computing (SOC) two key problems arise. Firstly, in the relatively open environment of SOC, services need to be composed that are heterogeneous, and possibly unknown, unreliable or/and un-trusted. However, architectures for autonomic systems usually assume that the systems are composed entirely of autonomic elements [8-10]. This approach presents difficulties in open systems where elements may have varying levels of autonomy and management capability. In addition, the composition does not necessarily have access to the internal behaviour of the component services that make up the

composite, and therefore cannot guarantee that the component has the desired autonomic properties [11].

A third broad approach to distributed management has been agent-based. In these approaches (e.g. [12]) agents typically represent independent services and resources and also use various techniques to plan, negotiate and enact workflows over those services. While such approaches are extremely flexible they may be too open ended for many types of domain. Entities are viewed as free agents rather than entities working within a constrained organisational structure such as a supply chain. This has led to agent languages (e.g. FIPA_ACL [13]) concentrating on providing semantics for negotiation and communicating beliefs, rather than management.

Finally, management standards have been recently developed that focus on Web services. WSDM-MUWS (WS-Distributed Management – Management using Web Services) is an OASIS specification [14, 15] for web service based management. MUWS provides an XML schema for creating manageability endpoints for a resource which are exposed as WSDL descriptions of a web service. The values which can be manipulated by a resource controller are presented as resource properties in a resource properties document. These resource properties can be queried, updated, or subscribed to for notifications of changes, making MUWS another operational management standard. MUWS is becoming a popular web service based management standard and has various implementations, most notable of which is Apache Muse [16]. WS-Management [1] is a standard by the DMTF and is a competitor standard to WSDM / MUWS. WS-Management seeks to be a smaller, simpler standard. The standard defines a minimal set of specifications and usage requirements, but leaves it to the developer to customise needs for the specific resource being managed. The most notable implementation of WS-Management is Wiseman [17].

MUWS and WS-Management attempt to provide a standard way to represent management interfaces which allow the separation of management related operations from normal functional operations. This allows the managers / operators of resources with management interfaces to change configurations, diagnose problems, and conduct monitoring. The manager of the resource is typically also the consumer of the resource. In this situation the manager and the consumer exist within the same administrative domain as illustrated in Figure 1. Consumers outside the administrative domain typically do not have access to management interfaces, just as a remote user of a system in an organisation does not typically have administrative rights to that system.

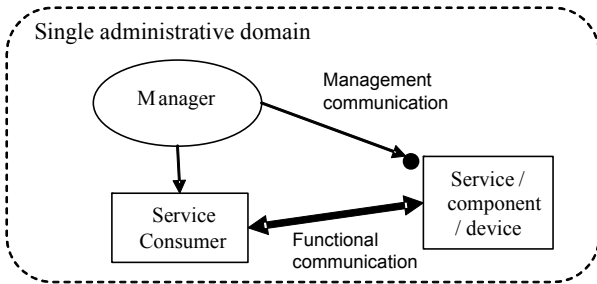


Figure 1: Managers and consumers in a single administrative domain

In open systems there may be situations where it is desirable for a consumer outside an administrative domain to have access to the management interface of a service. The consumer could then fine tune the operation of the resource to suit their own requirements (Figure 2).

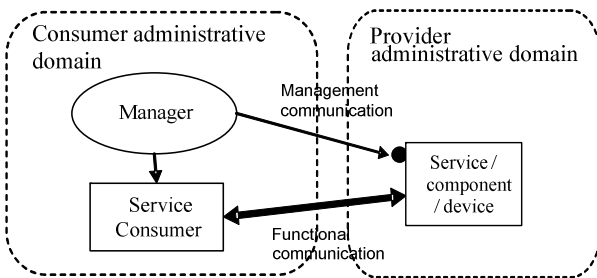


Figure 2: Managers and consumers in multiple administrative domains

If a consumer is not allowed the right to access these configuration options, there should be the possibility of requesting a change to the entity with management rights within the services own administrative domain. This is the case in self-managed systems, where the service may possess some internal manager (Figure 3). In this case the managers need to coordinate their activities and come to some agreement on what will be provided to ensure quality of service.

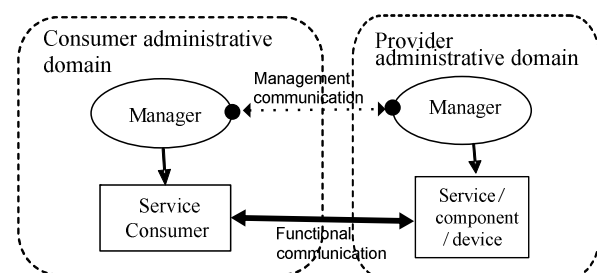


Figure 3: Communication between managers in multiple administrative domains

Current management standards could potentially be used to support the second case where a consumer outside the services administrative domain has some level of management access to the resource. This could be achieved by exposing a public version of the management interface. They do not however address

the third case of managers communicating across administrative domains to coordinate the interoperability of their services. This requires operations outside the scope of MUWS and WS-Management. In this paper we propose an extension to WS-Management that addresses all the above cases. WS-Management is a suitable choice due to its simplicity and the fact it has been designed with customization in mind. In cross administrative domain communication there is a need to manage various quality aspects of a management relationship, such as verifying that the party of the other domain is trustworthy before allowing any functional communication to take place. We do not attempt to address these problems here, we solely focus on the operations and message types required to facilitate cross administrative domain management communication.

3. Motivating scenario

Vehicle interaction using wireless networks has been identified as a research challenge and a future growth area in the automotive industry [18] and provides a good example of the types of management relationships required between two independent systems / services. In our example we will use a scenario concerning two cars, each equipped with software systems which need to interact with each other. Each car is an autonomous entity and neither has any authority or control over the other. This situation maps to the third case discussed in section 2, where each cars system has its own manager and its own administrative domain.

Jane and Bob are the drivers of two cars travelling from Melbourne to Sydney. The telematics systems of the two cars allow cooperative route planning and for them to form a convoy. That is, one car searches the best route based on the group's preferences and the other car follows exactly the same route. The telematics system of the leading car (driven by Jane) is set to a "leader" mode, which allows re-calculating the route during the travel. And, the telematics system of the following car (driven by Bob) is set to a "follower" mode, which ensures that the following car takes the same route as the leading car (instead of searching a travel route itself which could potentially differ).

The route planning systems have a number of constraints. First, the following car needs to send to the leading car an update of the distance between the two cars every 10 seconds. Second, when the distance between two cars is more than 300m, a warning message is presented to the drivers of both cars. Third, if any of the two cars stop or experiences a mechanical warning or failure, a notification message is sent to the other car.

During the trip, the weather turns bad and it starts raining heavily. The telematics system of Jane's car sends a suggestion to Bob's car to change the desired travel distance from 300m down to a shorter distance,

say 100m, as the heavy rain could affect the quality of communication between the two cars.

When the group is near Sydney, the battery of the following car starts running flat. From the car's Electronic Control Unit, the telematics system knows that this is a serious issue owing to the failure of an alternator. The car is only drivable for another 5-10 minutes. The telematics system sends a notification to the leading car to alert them of the failure and to request assistance.

In the above example, the systems interact with each other to achieve their desired goals in a peer-to-peer arrangement, no car's system has control over the other - they merely work together. The current management standards discussed in section 2 such as WSDM and WS-Management do not provide the means to deal with this scenario. While they could be used to allow one car's system to invoke management related operations on the other for getting and setting values such as the current distance update interval, they do not cater for the fact that these are independent systems each in their own administrative domains. A car's telematics system has no right to simply approach another and start invoking operations such as this and it certainly does not have the right to reconfigure any values. The telematics systems require a way to communicate with other cars in such a way that they can make proposals and requests, and establish agreements which set out the terms of their convoy. In our case an agreement is required between the two cars that defines the obligations that the leader and follower cars have to one another, such as distance updates and route planning information.

4. Facets of management

Management interaction between independent services can have a number of facets that are determined by the respective management capabilities of the services. What level of capability the services present, and whether they are respectively a service consumer or provider then determines the types of messages that are exchanged between those services.

4.1 Levels of management capability

If services are to inter-operate, there exists the possibility that these services may come from a wide variety of application domains. They may have been designed with different circumstances in mind and therefore may possess a differing level of capability. They may also exist in different administrative domains and have different owners / controllers. The ability for a service to allow runtime management of its functionality and the ability to adapt to changes required by its relationships will have an impact on the relationships it may participate in. For example in our car scenario, if the follower car is not capable of accepting manageability requests from the leader car,

they may not be able to participate in the convoy. We identify three levels of runtime capability to allow manageability a service may exhibit:

1. *No management capability*: Basic services which only allow functional communication fall into this category. For example a web-service that validates emails provides a WSDL description of its functional interface. The service provides no means to change the functionality of the service or customise the way in which it validates emails.
2. *Operational management capability*: We say a service that provides a way to configure the functionality it provides has operational management capability, i.e. the service provides a set type of functionality but it allows for configuration of the way it provides that functionality or the non functional requirements associated with it. In our scenario, the follower car sends distance updates every 10 seconds. If it allowed the leader car reconfigure this to 20 seconds via some exposed management operation, this is operational management.
3. *Contractual management capability*: The most advanced form of management capability a service may possess is contractual management. We said operational management capability allowed for an external entity to make configuration changes to the service. In the case of contractual management a service has the ability to adjust the functions it provides or invokes beyond the pre-configured options. The service possesses some self-management ability, meaning configuration changes originate from within the service as opposed to externally. This means any input from an external entity must take the form of a request. In order to achieve interoperability a service with contractual management capability forms an agreement with external entities. For example, in our scenario both cars telematics systems are self-managed. If the leader car wishes the follower car to send distance updates at 20 seconds intervals as opposed to 10 second intervals, it must request this of the follower car and make some amendment to their previously established agreement. The follower car may or may not accept this request, but its decision to do so may affect its relationship with the leader car, i.e. the leader car may decide to stop transmitting route information, effectively terminating their agreement.

Depending on the capability a service possesses, it may participate in relationships that require it to take the role of a lesser capability. A service may deprecate its management capability if that capability is not required by its interactions (Figure 4). For example, a service with a range of operating modes (i.e. with operational management capability) may restrict its appearance to that of a basic service (i.e. a service with no management capability) if only one mode is required in that service context. Likewise, a service with

contractual management capability may behave as if it only has an operational interface if that is all that is required of the situation.

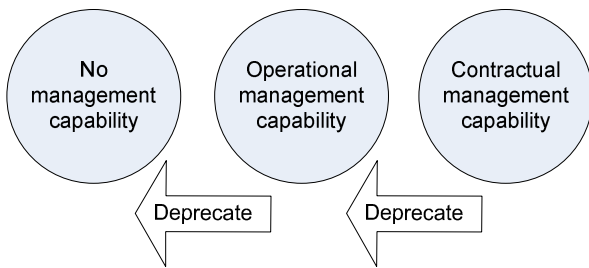


Figure 4: Levels of management capability and deprecation

The three levels of capability differ when considered from the perspective of a provider and the perspective of a consumer. In the case of a service with operational management capability the service exposes operations to allow an external entity to make configuration changes. This is made use of when the service is acting as a service provider, with the consumer acting as a manager invoking the exposed operations.

On the other hand, a service with contractual management capability is able to act as an agreement initiator or a receiver, meaning the service must expose the contractual operations as well as have the ability to invoke them on others. In order for the service to be capable of entering into an agreement with other parties, it must be capable of receiving and processing agreement related messages such as proposals, but also capable to making its own proposals / counter proposals. A service with contractual capability is able to enter into a broader range of relationship types that are more peer-to-peer oriented. A service with operational capability may only be able to enter into more restricted forms relationships where the service takes a role that involves having management operations invoked on it, not vice versa.

4.3 Management messaging requirements

Each level of management capability has requirements for different forms of messages. A service with no management capability has no manageability requirements. It simply needs to expose its functionality, e.g. in the case of a web service, via its WSDL interface. Operational and contractual capabilities are described below.

Operational management messages. The messaging requirements of operational management dictate it must allow the ability to customise its operational configuration and to allow the querying of current configuration properties. We define three message types required in line with existing management standards such as WS-Management:

- *Queries:* Queries on current configuration properties. E.g. the current send interval of distance updates.
- *Updates:* Updates / changes to configuration properties. Generally any value that can be updated may also be queried.
- *Subscriptions:* Subscriptions to topics of interest to allow asynchronous notifications of events. E.g. the follower cars distance to the leader becoming greater than 200m. Subscriptions allow a service to effectively monitor the state of another.

Operational management capability is generally exhibited by services that make use of the existing management standards described in section 2. These three message types are typical in all operational management standards.

Contractual management messages. Contractual management is based on the presumption a service is self-managed and will need to create and abide by agreements in order to form a relationship with another service. With this in mind we define the following messaging requirements:

- *Agreement discovery:* A service needs to be able to obtain a set of acceptable agreements from any service it wishes to form an agreement with. This provides a base on which to build a new agreement.
- *Agreement formation:* The ability to send / accept proposals from another party and respond to those changes with acceptance, rejection, or a counter proposal.
- *Agreement commencement:* The ability to notify and accept notification that a pre-defined time or date has been reached and a previously formatted agreement is due to comment.
- *Agreement maintenance:* The ability to renegotiate an existing agreement and the ability to notify / accept notifications from another party that an agreement term has been breached, effecting the state of an agreement and possibly resulting in the invoking of an agreement clause in response.
- *Agreement termination:* The ability to exit an agreement and notify the other party.

The messages and operations exposed in regard to the above requirements must allow a wide variety of domain specific negotiation protocols. While we define the message types and interface operations needed for such protocols, the definition of these negotiation protocols is outside the scope of this paper. The representation of the agreements themselves is also not in the scope of this paper but current standards for this exist such as WS-Agreement [19]. It is the contractual capability described here that we implement as a part of our multi faced management interface.

5. Management interface definition

Given the above messaging requirements for services with various management capabilities, we now define a management interface that can present zero or more facets suitable to the type of management communication needed.

A service with no management capability presents a normal WSDL functional interface. This is an accepted standard and allows consumers to bind to the service using commonly accepted standards and messaging protocols. For services with operational capability web service based management standards discussed in section 2 provide the necessary operations. However, for services with contractual capability there is no common standard for a management interface. We propose a management interface that extends on the operations exposed by current standards to address operational and contractual capability in a single interface, allowing independent services of differing capability to easily enter into a relationship and manage that relationship.

An overview of the structure of the management interface for a service with contractual and operational management capability is shown in Figure 5. Using a single management interface regardless of capability allows creation of a standard management channel between two services to accompany the standard functional channel.

A management interface may include an operational capability facet, a contractual capability facet, or both. This allows a service with greater capability to deprecate to a lower one as previously discussed. A service with contractual capability may also wish to expose limited operational capability to other services with which it is in a relationship.

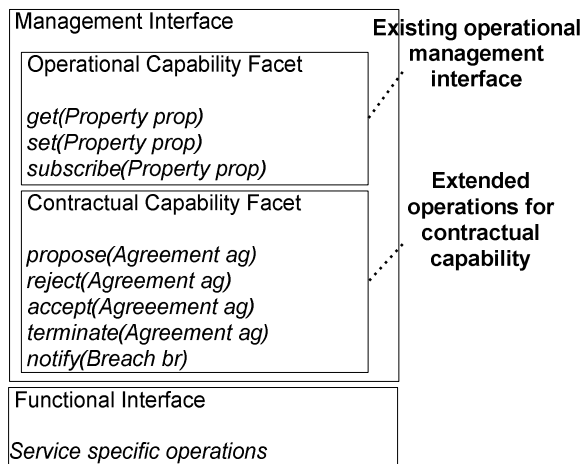


Figure 5 Structure of management interface

Each section in the management interface contains operations relevant to its level of capability:

- Operational Capability contains any operations allowing the retrieval, update / change, and

subscription to any properties the service wishes to expose. If a service also has contractual capability then the updating / changing of any properties will not be present unless explicitly made available.

- Contractual Capability operations must support the allowable communication we define as contractual management capability. For example agreement operations for agreement discovery and agreement formation.

In order to make the management interface as easily accessible as possible it makes sense to extend a current standard. WS-Management seeks to achieve the following capabilities:

- Get, put (update), create, and delete individual resource values.
- Enumerate the contents of containers and collections.
- Subscribe to events emitted by resources.
- Execute specific management methods with strongly typed input and output parameters.

The last point is important as it gives us scope to extend WS-Management with operations that allow for contractual management capability.

For example in our car scenario, both cars may possess contractual capability and have entered into a relationship stating their required actions (send route data, distance updates, etc). The follower car may wish to expose readily available information to the leader as an operational query, such as fuel levels.

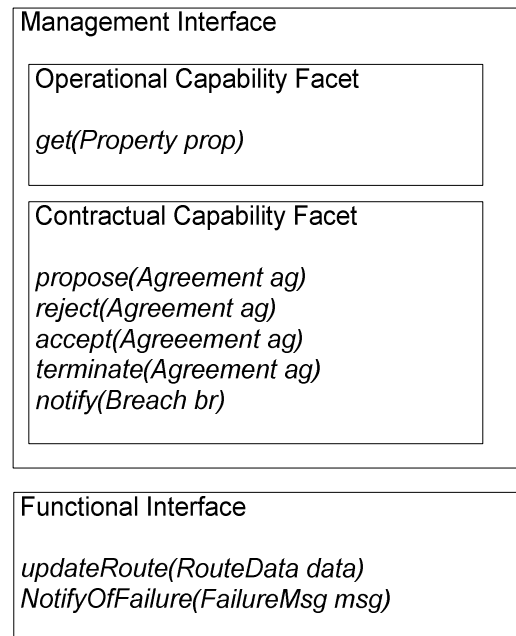


Figure 6: Structure of a management interface for the follower car

We define a new WSDL port type which contains the operations illustrated in the contractual capability

section of Figure 6. Operations for receiving route updates from the follower car or notifications that the follower car has experienced a technical difficulty reside in the functional interface.

Port type overview

```
<wsdl:portType name="ContractManagementPort ">
  <wsdl:operation name="getAllowableAgreements">
    <wsdl:input message="tns:AARequest"></wsdl:input>
    <wsdl:output message="tns:AAResponse"></wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="propose">
    <wsdl:input message="tns:agreement"></wsdl:input>
  </wsdl:operation>
  <wsdl:operation name="accept">
    <wsdl:input message="tns:agreement"></wsdl:input>
  </wsdl:operation>
  <wsdl:operation name="reject">
    <wsdl:input message="tns:agreement"></wsdl:input>
  </wsdl:operation>
  <wsdl:operation name="terminate">
    <wsdl:input message="tns:agreement"></wsdl:input>
  </wsdl:operation>
  <wsdl:operation name="notify">
    <wsdl:input message="tns:breach"></wsdl:input>
  </wsdl:operation>
</wsdl:portType>
```

Proxies can then be generated for use by other parties

Example Proxy Pseudo Code

```
Interface FollowingCarProxy {
  AAResponse getAllowableAgreements()
  void propose(Agreement ag)
  void accept(Agreement ag)
  void reject(Agreement ag)
  void terminate(Agreement ag)
  void notify(Breach br)
}
```

An AAResponse message (allowable agreement response) contains an array of agreement templates. A propose() has an agreement parameter being proposed to the other party. This agreement may or may not be accepted by using the accept operation. Operations also exist for rejecting proposals and counter proposals.

5.1 Incorporation into the ROAD framework

In order to put the management interface we have defined to use, we need to incorporate it into a framework that provides a way to represent external agreements as contracts, and manages the interdependencies between those contracts. For this purpose we make use of Role Oriented Adaptive Design (ROAD) [2]. ROAD is a meta-model for designing self-managed systems. Using ROAD, services are designed as a collection of roles. A role is a position description within the system and is a separate runtime entity from whatever component plays that role. Players of roles can potentially range from a number of runtime entities,

such as agents, web services, objects, human operators, or other self-managed composites. Roles are bound to each other by contracts. Contracts define the allowable interactions between roles (and in turn players) and also define the obligations roles have to one another. The requirements expressed by a role (the position description) are the aggregate of all the connected contracts. Collections of roles and contracts are organised into self-managed composites. A composite is managed by an organiser. The organiser is the management mechanism in ROAD. It is responsible for altering performance requirements on existing terms of contracts and monitoring for breaches. It is also capable of interchanging players of roles at runtime. An example ROAD composite is shown in Figure 7.

If in ROAD self-managed composites play roles within each other, along with role players with more basic capabilities, a management interface will be required for players to allow the composite organisers to manage the different levels of relationships that may occur between one and other.

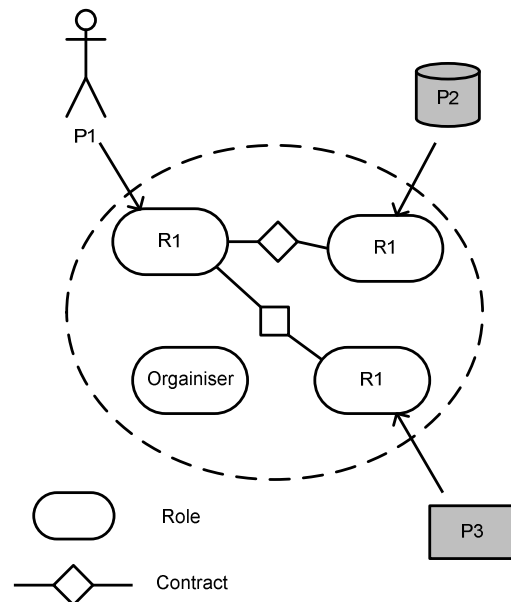


Figure 7: Example ROAD composite

As role players in a ROAD composite vary, we define three levels of players to match our three levels of management capability:

1. Basic player. A basic player has no management capability and simply provides a service.
2. Management enhanced player. A player that has operational management capability.
3. Self-managed player. An autonomous player that has contractual management capability.

The ROAD composite, being a self-managed entity, binds to the services provided by basic and management enhanced players. In the case of a management enhanced player, it is the composites organiser who can

make use of the players management interface and configure the player via the exposed operations.

The case of a composite playing a role inside another composite is of particular interest to us as it demonstrates the need for contractual management. In this situation both composites are self-managed, meaning neither has control over the other. The composites will be required to coordinate with each other to achieve their goals. We say that a composite playing a role inside another needs a representation of itself in that composite. In the case of our car example, if both Bob's and Jane's cars were self-managed composites; each has a role representing the other in their own respective structures. The agreement binding the two cars can then be represented as a contract inside both the composites. If the composites organiser wishes to alter the internal contract, it must propose this to the other car's composite via its management interface. If the contract is related to a role being played by a service with only operational capability, it may simply be able to reconfigure the service when altering the contract.

In Figure 8, we see both cars composites have a representation of the other in their own structures. Either car can now interact with the other functionally, via the appropriate role. The agreement formed between the two composites is represented as contracts. In Figure 8 the GPS systems which are responsible for keeping track of distance between the cars has a contract with the car roles that requires it to send updates at a certain interval.

In order to achieve this kind of interoperability, both composites will need to expose a management interface to facilitate the creation of contracts. The management interface in this case acts as the endpoints of a

communication link between the composites organisers we call this link the *management channel*.

In Figure 9 we illustrate a ROAD composite with examples players of all three of our player types. With a standard management interface in place the composites organiser can easily facilitate binding with players regardless of their capability.

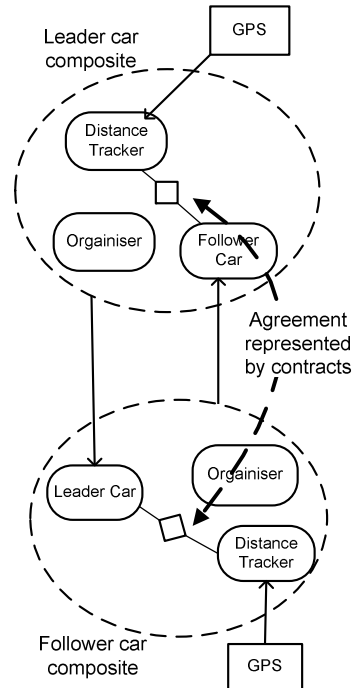


Figure 8: ROAD composite role playing

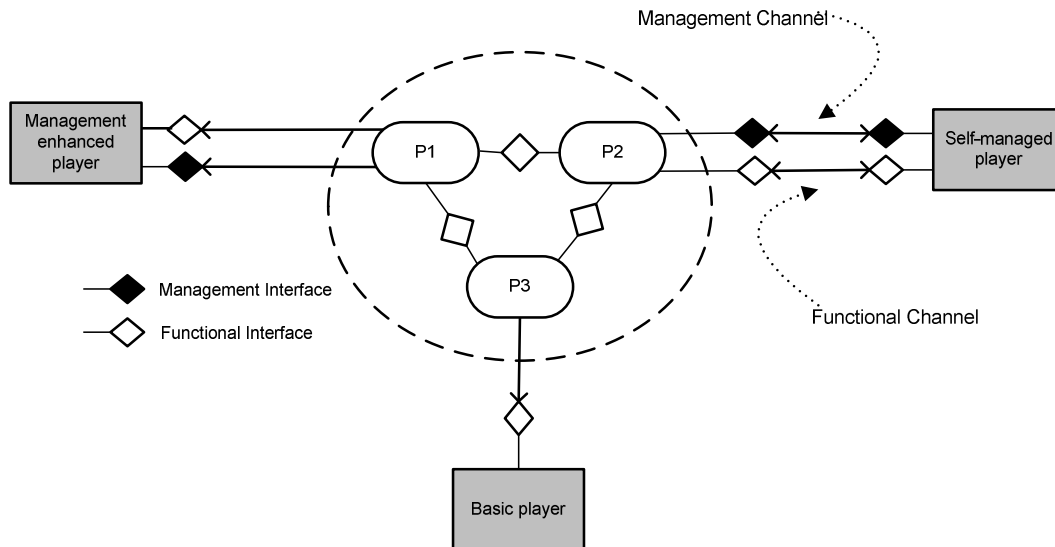


Figure 9: The three player types and their management interfaces in a ROAD composite

6. Conclusion

In this paper we propose a multi faceted management interface to facilitate interoperability between independent services in different administrative domains which have varying levels of capability and adaptivity. The proposed management interface categorises services into three facets; no capability, operational capability, and contractual capability. Where no capability refers to a basic service with only a functional interface, operational refers to a service that allows configurability from an external consumer or controller via operations exposed by existing web service management standards, and contractual capability that refers to a service that can provide varying functionality depending on an established agreement. Standards exist that provide operational management of software systems. These standards do not adequately address the need for systems across administrative domains to exchange management related information and to facilitate the forming of agreements with one another.

We demonstrate the use of the management interface in a scenario where driving tasks are coordinated between cars possessing self-managed telematics systems. In this scenario each car possesses an independent autonomic system. No system has the administration rights to any other cars system, rendering existing management standards inadequate. We illustrate how a system such as this might be implemented using ROAD, a framework for developing self-managed composites of services that explicitly represents external agreements with services as contracts, and manages interdependencies between those contracts.

Implementation of the ROAD framework has been through various iterations as conceptual ideas have been developed. Currently development efforts focus on an application called ROADfactory. The purpose of ROADfactory is to instantiate a fully functioning ROAD composite from an XML file which defines the contracts and structure of the composite. A graphical tool known as ROADdesigner is also under development which allows a designer to define a composite using a drag and drop interface. An infrastructure to deploy composites as web services is planned as future work.

7. References

- [1] Arora, A., Cohen, J., Davis, J., Golovinsky, E., He, J., Hines, D., McCollum, R., Milenkovic, M., Montgomery, P., and Schlimmer, J.: 'Web Services for Management (WS-Management)', Distributed Management Task Force (DMTF), October 2004
- [2] Colman, A.: 'Role-Oriented Adaptive Design'. PhD Thesis, Swinburne University of Technology, Melbourne, Australia, 2006
- [3] Case, J., Fedor, M., Schoffstall, M., and Davin, C.: 'A Simple Network Management Protocol (SNMP)', The Internet Society, April 1999
- [4] Papazoglou, M.P.: 'Web Services: Principles and Technology' (Prentice Hall, 2008)
- [5] Common Information Model (CIM): <http://www.dmtf.org/standards/cim/>, accessed October 2008
- [6] Web Based Enterprise Management (WBEM): <http://www.dmtf.org/standards/wbem/>, accessed October 2008
- [7] Nami, M.R., and Bertels, K.: 'A Survey of Autonomic Computing Systems'. *Proc. Third International Conference on Autonomic and Autonomous Systems (ICAS)*, 2007
- [8] Kephart, J., and Chess, D.: 'The Vision of Autonomic Computing', *Computer*, 2003, 36, (1), pp. 41-50
- [9] Menasce, D.: 'QoS-aware software components', *Internet Computing, IEEE*, 2004, 8, (2), pp. 91-93
- [10] White, S., Hanson, J., Whalley, I., Chess, D., Kephart, J., Center, T., and IBM, W.: 'An architectural approach to autonomic computing'. *Proc. First International Conf on Autonomic Computing (ICAC'04)*, 2004
- [11] Colman, A.: 'Exogeneous Management in Autonomic Service Compositions', *Proc. of the Third International Conference on Autonomic and Autonomous Systems*, 2007
- [12] Chhetri, M., Mueller, I., Goh, S., and Kowalczyk, R.: 'ASAPM-An Agent-Based Framework for Adaptive Management of Composite Service Lifecycle', *Web Intelligence and Intelligent Agent Technology Workshops, 2007 IEEE/WIC/ACM International Conferences on*, 2007, pp. 444-448
- [13] FIPA.: 'ACL Message Structure Specification', Foundation for Intelligent Physical Agents, 2000
- [14] Bullard, V., AmberPoint, I., and Murray, B.: 'An Introduction to WSDM', OASIS, 2006
- [15] Bullard, V., and Vambenepe, W.: 'Web services distributed management: Management using Web services (MUWS 1.1) part 1', OASIS, 2006
- [16] MUSE: <http://ws.apache.org/muse/>, accessed October 2008
- [17] Wiseman: <https://wiseman.dev.java.net/>, accessed Oct 2008
- [18] Broy, M.: 'Challenges in automotive software engineering'. *Proc. 28th international conference on Software engineering*, 2006
- [19] Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Pruyne, J., Rofrano, J., Tuecke, S., and Xu, M.: 'Web Services Agreement Specification (WS-Agreement)', Open Grid Forum, March 2007