

# Exogenous Management in Autonomic Service Compositions

Alan Colman

Faculty of ICT, Swinburne University of Technology

PO Box 218, Hawthorn, 3122

Australia

acolman@ict.swin.edu.au

## Abstract

*This paper proposes that software architectures that support autonomic service-oriented computing need to have an exogenous management structure. Exogenous management regards autonomicity as a property of relationships between elements, rather than a property of the elements themselves. We explain the concept of exogenous management, and show how a number of desirable attributes that support autonomicity flow from this approach. These attributes include self-management; separability; recursive composition; and grounding through the monitoring of interactions. We will show how these attributes help enhance the adaptability and control the complexity of context-aware compositions of services. We then discuss how this exogenous approach to management has been implemented in the ROAD (Role-oriented Adaptive Design) programming framework. This framework is extended by software developers to create service compositions whose level of autonomicity can be incrementally modified at runtime.*

## 1. Introduction

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 untrusted. However, architectures for autonomic systems often assume that the systems are composed of autonomic elements [9,10,15]. This approach presents difficulties in open systems where 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.

A second major problem is how to control the complexity inherent in autonomic systems. An autonomic system has the ability to maintain its function in the face of changing user, computational

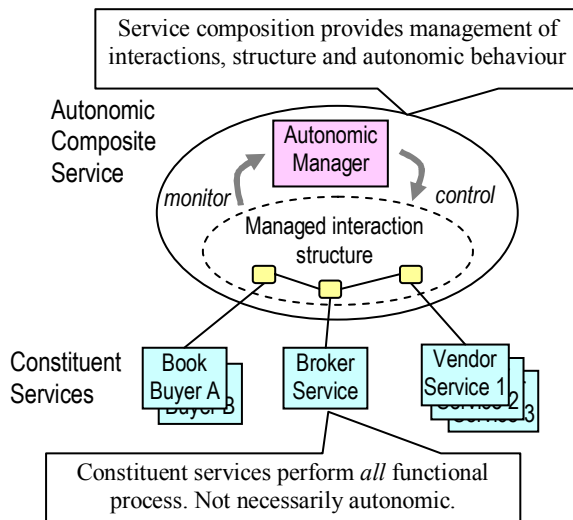
and network contexts. The autonomous system adapts itself to these environments by managing its own structure and behaviour: so called self-\* properties. In order to effectively manage itself, an autonomous system needs to be both self-aware and context-aware. For a system to be able to adapt to more than just a limited number of environmental states, it not only needs to have a reflective understanding of itself and its goals, but also needs to have a representation of its 'real world' context. In short, self-aware systems need to be *grounded*. In designing such systems, however, the problem arises that any representation of a context is likely to be incomplete. Environmental models are almost by definition partial models, given the complexity and multi-faceted nature of most real-world environments. While in simple systems it is possible to identify certain variables of interest that can be sensed, measured and controlled, as systems grow in size the interactions between such variables can become exponentially complex. If large scale autonomic systems are to become a reality, a way must be found to control this complexity. Autonomic architectures, as suggested by [7], that attempt to take account of various user, computational and network contexts in a generalised way are themselves very complex, and consequently may be impractical. What is needed is an approach that implements a level of autonomicity appropriate to the amount of volatility or uncertainty in the system's environment.

In this paper we present an architectural approach that does not assume the constituent services are necessarily autonomic, known or trusted, and that controls the complexity inherent in the building of autonomic systems. To achieve these goals, we propose that software architectures that support autonomic service-oriented computing need to have an *exogenous* management and structure.

*Exogenous management* is the strict separation of the structure and management of a composition from the functional processes that are performed by the services. This separation is only apparent if we look inside a composition. Viewed externally the composition is itself a service that expresses a function

like any other service. Compared to a basic service, an autonomic composite service will be able to adapt to a greater number of environmental states. In cybernetic terms the adaptability of the composite is a consequence of its greater *variety*, that is, its ability to express a relatively large number of internal states [1]. Figure 1 illustrates this separation of concerns. The key difference between an autonomic composite service (ACS) and an autonomic element, as proposed in [9], is that the ACS performs no domain function in itself. The autonomic manager of the ACS controls an *interaction structure* rather than directly managing any functional element.

It is commonplace in autonomic architectures to separate managers from the functional elements which those managers control. However, what distinguishes a strictly *exogenous management* architecture from other approaches is that the manager of compositional structure has no access to, or direct control over, the internal workings of the constituent services. The exogenous manager primarily controls *relationships*. It defines the compositional structure, and models the required performance across that composite structure by defining abstract service definitions (roles) for the *expected* external behaviour of services bound to the composition. It then binds services to those roles based on the services' *actual* measured performance (or a *claimed* performance if no history of performance is available). The manager then *monitors* and *controls* the runtime *interactions* over the composite structure.



**Figure 1. Exogenous management: Strict separation of management and structure from process.**

Figure 1 illustrates an exogenously managed composite that mediates between services that purchase, broker and sell books. These services initiate all the functional transactions related to the purchase of books: searching, quoting, selection, ordering, supplying and payment. The composite, on the other hand, stores the

functional and quality requirements of each of the services (for example, the Book Buyer and Vendor what both have terms-of-trade they require), and measures the conformance of the constituent services to those requirements. The composite manager's task is to regulate and configure the composite in order to find an optimal solution to the (possibly inconsistent) requirements of the various services, given their measured or claimed performance.

Just as the constituent services are opaque to the composite, so too is the internal structure of the composite invisible to these services. The constituent services view the composite as just-another-service with which it has mutual obligations. All interactions between constituent services are controlled by the composite, and constituent services are unaware of each other's existence.

The structure of this paper is as follows. In the next section we show that exogenous management is consistent with a number of architectural attributes that support autonomicity of service compositions. Section 3 demonstrates how these attributes can be implemented using the Role-oriented Adaptive Design (ROAD) framework. Section 4 discusses how ROAD service composites at different levels of abstraction can be incrementally added to composition to provide the required level of autonomicity, and Section 5 concludes.

## 2. How exogenous management supports autonomicity

Systems with exogenous management have a number of inter-related architectural attributes that facilitate autonomicity in service compositions. These attributes are as follows: self-management; separability; recursive composition; and comprehensive grounding. In this section we will explain what we mean by these terms and show how they support autonomicity.

**Self-management** is a *sine qua non* for autonomic computing. Self-management is seen as the answer to controlling the "complexity crisis" as computer systems proliferate and become more complex and interconnected [9]. An exogenous management system can be viewed as the ultimate expression of self-management because it only manages itself (the set of relationships it controls) and *nothing else*. Recall, from the definition of an exogenous management structure above, that all composites and services (even the basic services that are bound to the composite) are mutually opaque. In addition, *all* functional processes are encapsulated in services. While the manager of the composite controls the *configuration* of the composite, the only direct *regulatory* management is the management of interactions that flow through the composite. In other words, the manager of the enclosing composite only controls the *relationships*

between the services composed by the composite. Although the manager of the composite may request that its constituent services fulfil certain quality requirements (if those services allow), and it may select services accordingly, it has no access or management control over the internal relationships within those constituent services. The manager of the composite is not a master controller trying to “micro-manage” its constituent services. All management *between* composite services involves the passing of declarative quality requirement requests.

Although such an approach restricts the power of the composite manager, it accords well with the openness of the SOC paradigm and autonomy of services. This approach also helps manage the complexity of service compositions. Each composite only has to manage its own set of *relations* between services bound to the composite. It does not control the services themselves – it merely specifies what behaviour the service must exhibit if it is to participate in the composition. While it would, of course, be possible to create a complex composite with a large number of interrelated services, organisational theory [11] posits that the systems are best decomposed in such a way that the management of any one composite should have a limited span of control. As autonomic managers are, for the foreseeable future, likely to have “limited rationality”[14], it follows that a composite should only manage a relatively small set of relationships at a single level of abstraction.

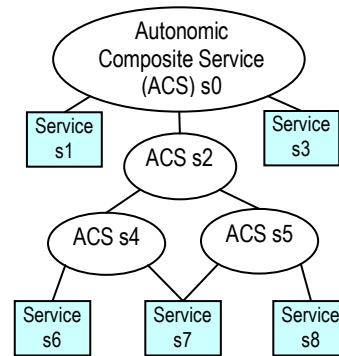
As each composite maintains its own set of relationships with all its constituents, and controls all interactions through connections according to its requirements, exogenous management provides a high level of self-protection for the composite.

**Separability.** One of the key attributes of an autonomic system is that of *self-configuration*. Self-configuration has two dimensions in an exogenously managed composite, and indeed in most autonomic architectures. Firstly, autonomic compositions must be able to alter the structure of their internal relationships. Secondly the manager of a composite must be able to select the external services to be bound to the composite, and control the binding to those services. In particular, the ability to *separate* the composite from its constituent elements is an essential feature of exogenous management. A consequence of a composite having no management rights over its constituent services (whether composite or basic) is that the performance of those services, with respect to the composite, cannot be guaranteed. For a composed system to exhibit autonomic behaviour at the system level, and yet have some of its elements that are possibly non-autonomic, it must be able to replace elements within the system that are not meeting their required function or quality requirements. (And even if all constituent services are autonomic, autonomicity is

never an absolute guarantee of performance or behaviour.)

A number of properties are required of an architecture if the integrity of the composite is to be preserved during service separation. Firstly, the structure (internal relationships) of the composite needs to be preserved, along with the abstract service definitions that define the requirements of constituent services. These abstract service definitions therefore need to be runtime entities rather than just design specifications. Secondly, the composite still needs to function as a service while a constituent service is being swapped, or in the event of failure or underperformance of a constituent service. The composite therefore needs to provide buffering for messages that flow through the internal structure of the composite. Finally, the mutual opacity of the composite and its services means that all information on the structure of the composite must be maintained by the composite itself. In other words, services in the composite do not hold direct references to each other. It is the composite’s responsibility to provide content-based routing so that services can communicate with each other.

**Recursive composition of opaque elements.** As an autonomic composite appears to be just-another-service, autonomic composites can be recursively composed.



**Figure 2. Recursive composition and limited span of control**

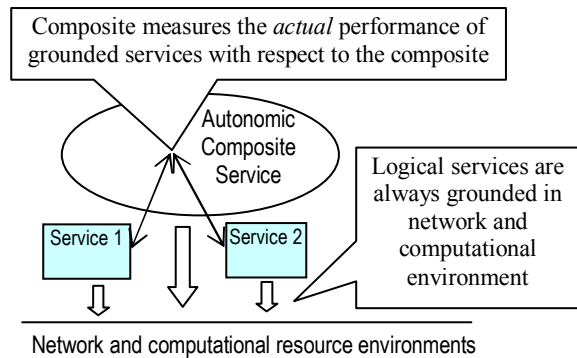
Figure 2 illustrates an autonomic service composite s0 that composes a number of services: the basic services s1, s3, and the composite service s2. Composite service s2 in turn composes a number of other services: s0, s4 and s5. Note that in this view composition is not a hierarchical concept. For example, composites s0 and s2 have each other as constituent services.

The separability inherent in an exogenous management approach makes it possible to compose simple sets of relations, with each set of relations at a single level of abstraction. Composites at higher levels of abstraction are then operationalised (decomposed) by composites at lower levels of abstraction. The process continues recursively until all composites ultimately are instantiated by basic services. Such an approach allows

the system designer to maintain a high degree of modularity and decoupling within the system. The complexity of lower level composites is hidden from composites at higher levels, and vice versa. As we will illustrate below in Section 4, recursion also simplifies the developer's task, in that the same meta-modelling constructs can be applied at all levels of abstraction.

**Grounded Services.** Much of the research into autonomic computing has focused on the development of applications and elements that are aware of their computational and network context: an awareness that enables them to adapt to perturbation in their underlying infrastructure. This is typically done by instrumenting various aspects of the infrastructure (e.g. memory, bandwidth, etc.) in order to model the potential impact on performance (e.g. [2,7]).

Work on service composition, on the other hand, has not adequately addressed the grounded nature of services. Grounded services always have a set of quality attributes associated with their actual behaviour in a particular context (e.g. response time, reliability, cost). Where quality attributes have attempted to be modelled in service compositions, it is generally based on an abstract characterisation of qualities the services *claim* to possess irrespective of their actual context (e.g. capacity of network connection from service to the composition; actual demand on the server etc.). What ultimately matters to a service composite is the *externally observed* (actual) behaviour of a service *with respect to* the composition as shown in Figure 3.



**Figure 3. Exogenous measurement of grounded service behaviour with respect to a composition**

Observed behaviour is *behaviour in relation* to other entities, and can be defined as a property of those relationships rather than an intrinsic property of the service itself. The quality of a service's behaviour is intrinsic if, and only if, it is invariant with respect to all other services in every possible context. An autonomous composite therefore needs mechanisms to measure the actual performance of a service with respect to the composition.

Because exogenous management is focused on the external behaviour of services which it models via their interactions, it provides a natural way to measure the

actual performance of grounded services. Indeed, the opacity of services means that it may not be possible to internally instrument the services or their infrastructure, and the open nature of the SOC environment means we can not assume services will expose the appropriate monitoring interfaces. That being said, it may be desirable for a composite to model some variable in its environment (e.g. network connectivity). In Section 4, we show how this can be done exogenously.

In summary, autonomicity is, strictly speaking, a property of the relationships between entities within a composite, rather than a property of the entities themselves. Of course, a composite (viewed externally as an entity) will express some level of autonomicity. But again, this autonomicity is always *in relation* to the context in which that composite is deployed.

### 3. Role-Oriented Adaptive Design: using roles and contracts to implement autonomic composites

The Role-Oriented Adaptive Design (ROAD) [4,5] provides a framework for the construction of exogenously managed autonomic composites as discussed above. ROAD consists of a few fundamental concepts: *roles* (which are played by services), *contracts* (which are connectors between roles) and *self-managed composites* that have internal *organisers*.

In ROAD, *functional roles* are first-class runtime entities that hold abstract service definitions. Because the services that play instances of roles can be transitory (for example, there may be no service currently available to play a role), roles store incoming message in queues. Roles also perform the function of message routers, as the services that are bound to a role in an organisational structure do not directly reference each other. Roles may be contracted to a number of other roles, and therefore need to forward messages from their service-player to the appropriate associated role.

*Contracts* perform three functions in a ROAD composite role structure: composition, interaction control, and performance monitoring. ROAD contract instances are dynamic and rich connectors between roles. By creating and/or revoking contracts, the topology of the composition of roles can be altered. As all roles (as opposed to services) are internal to the organisation, ROAD contracts are also internal to the organisation. All runtime communication between functional services bound to the organisation is *via* contracted roles, and, if necessary, contracts intercept the communications between roles. In this way contracts perform a similar function to interceptors in conventional middleware. Contract terms define the mutual obligations of the participant roles in an organisational context. They define the interactions that are permissible or required by the participant roles, and can be used to enforce sequences of interactions.

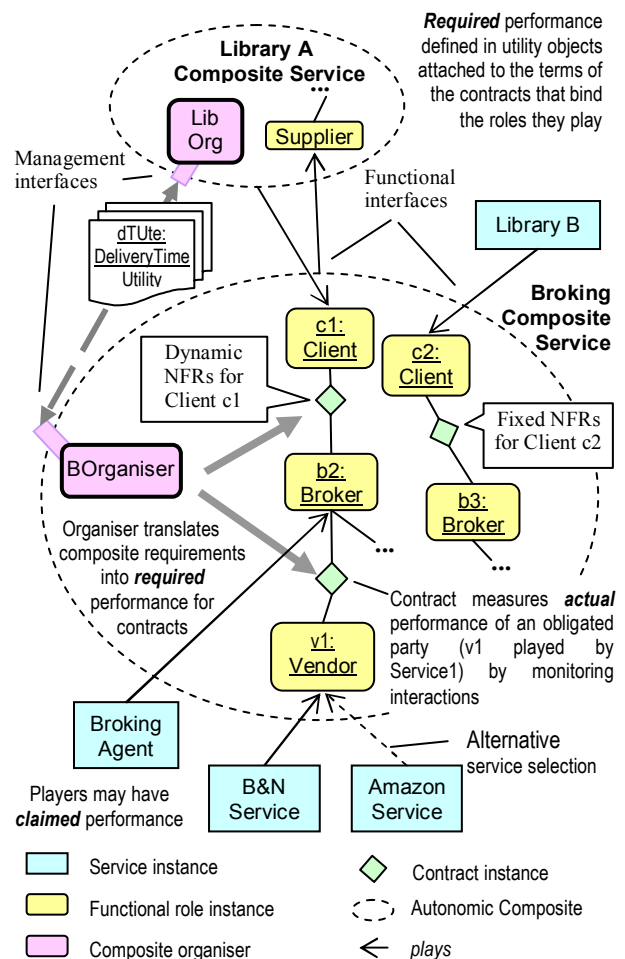
Contracts can also set arbitrary non-functional requirements (NFRs) in the form of utility objects, on their roles' interactions, and monitor those interactions for compliance to those requirements. In ROAD, contracts are implemented using association aspects [4,13], an extension to the AspectJ compiler. Such association aspects allow contract instances to be created that associate groups of objects, and can be used to intercept invocations between those objects.

*Organisers* are autonomic managers that create and destroy roles. They also make and break the bindings between composite roles and services (service selection), and create and revoke the contracts between the roles. They can thereby create various configurations of roles and services. Organisers set performance requirements for the contracts they control, and receive performance information from those contracts. Organisers have reconfiguration strategies they can employ if they detect under-performance in the composite they control. In short, organisers provide the autonomic behaviour of the composite application by managing the composition and instantiation.

Each organiser is responsible for the configuration of a set of roles and contracts: a *self-managed composite*. In terms of a management analogy, a self-managed composite in a business organisation would be a department (e.g. manufacturing department). An *instantiated composite* (i.e. the roles internal to the composite have services attached) can itself be a service. Messages to the composite are delegated to its internal role-players. An application composed of these self-managed composites can therefore be distributed (as any service can be distributed). As a composite can itself be a service that plays a role in another composite, it follows that composites can also be recursively composed/decomposed.

This ability to compose/decompose role composites enables complex systems to be modelled with a few conceptual constructs. Figure 4 below illustrates how these ROAD constructs can be used to create service composites. The broking composite mediates between purchases and suppliers of books. The Broking Composite Service instance is associated with two types of library. The first type of library service (LibraryA) is itself a composite service (not all of its internal roles or associated services are shown in the diagram). LibraryA plays the Client1 role in the broking composite. Conversely, the broking composite plays the Supplier role in the LibraryA composite. The implication of this bi-directional role-playing is that all functional messages between the composites pass through these respective roles. Such a structure would suit composites in different organisational domains (e.g. with different owners), because each composite has an internal role that is a proxy for the other composite. Functional coupling is thus kept to a single interface.

Each self-managed composite has exactly one organiser role. A self-managed composite has a management interface which is the external interface of its organiser role, as shown in Figure 4. This interface performs a similar function to an 'out-of-band' manageability interface in MoWS [12], in that required and actual performance-measures pass backward and forward over it. In the example below, Library A wants to reduce the maximum delivery time allowable for books it has ordered. Its organiser (LibOrg) sends the new quality requirement over the management interface to the organiser of the broking composite (BOrganiser). BOrganiser then writes the performance requirement into the appropriate contracts. This in turn may require the organiser to renegotiate with the services playing the contracted roles, or to replace those services with services that can meet the requirements (e.g. B&N is replaced with the Amazon service that can meet the new NFR.).



**Figure 4: Composition of autonomic composites**

The second type of library (LibraryB) in Figure 4 is not a role composite but is a basic service. It has a

functionally compatible service interface through which it interacts with the broking composite, but it has no management interface or organiser. While both LibraryA and LibraryB can play roles in the broking composite (i.e. order books from the broker), basic services like LibraryB cannot dynamically pass non-functional requirements (NFRs) to the composite. LibraryB's NFRs would need to be set statically in advance (or through some form of supervisory control via BOrganiser). A more detailed discussion of the transmission of NFRs between ROAD service composites can be found in [3].

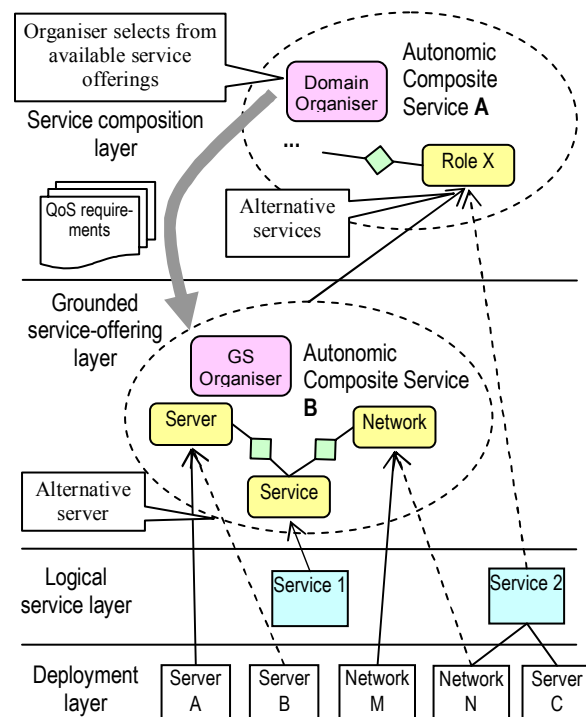
#### 4. Modelling grounded services as autonomic composites

As discussed in Section 2, a grounded service is a service running within a particular network and computational context. It is therefore a combination of a functional software component and the infrastructure on which the logical service runs and communicates. Grounded services are characterised both by a functional description and by a set of quality attributes (QoS, security, cost, etc.) in relation to the composite to which they are bound. The autonomic composites in the example outlined in the previous section can measure the observed actual performance of services (say, Book Seller services) with respect to their composite. In this case, if a service is underperforming, the only option open to the organiser of the Broker Composite Service is to *select* an alternative service that might better meet the requirements of the composite. For example, in Figure 4 the organiser could select the Amazon service rather than the B&N service.

However, in certain situations observed performance at this high level of abstraction may be too crude a measure to enable the organiser to make an informed decision. For instance, the problem with the QoS from the B&N service may be a result of a slow or unreliable network connection between the service and the composite, rather than being a result of any problem with the B&N service itself. Particularly in situations where the real-time performance of the composite is critical, or where the service code is distributed across a number of servers, the ability to model the infrastructure on which the composite relies becomes important. At these lower levels of abstraction, the elements in the infrastructure can be modelled using the same ROAD meta-constructs that are used to model the higher level application-domain.

Figure 5 illustrates how a ROAD autonomic service composite can be used to model a grounded service. The ground service (ASC B) has roles that represent the mutual requirements of the logical service, the server and the network. The roles that represent the server and network resources in a composite are similar to the concept of 'virtual services' and resources [6,8]. The organiser of the grounded service (GSOrganiser)

receives QoS requirements (e.g. information throughput) from the organiser of Composite A, and translates these into NFRs appropriate to an infrastructure model (e.g. bandwidth, reliability, latency etc.). These derived requirements are then written into the Service-Server and Service-Network contracts. GSOrganiser needs to produce a configuration of these resources that meets the requirements imposed on ACS B, as a consequence of ACS B playing Role X in ACS A. In this case, the GSOrganiser has a range of servers and network options available from which it can select: it can choose between Server A or B, and between Network M or N. Furthermore, suppose it proves necessary to model the network itself because the network consists of some value-added services running on a connection service. To model this refinement a composite at a lower level (not shown) could be recursively decomposed to play the network role in ACS B. Indeed, this new composite could be added to the composition at runtime because it is seen as just-another-service.



**Figure 5. Service instantiation with a composite that models a grounded service**

Note that in Figure 5, from the top level composite's point of view the alternative players of Role X (ACS B and Service 2 respectively) both offer the same functional behaviour. The advantage of ACS B over Service 2 is that it can dynamically adapt to broader range of QoS requirements because it can reconfigure its infrastructure. The complexity of this adaptability

however is hidden from the client composite (ASC A). If such adaptability is required it can always be added as needed to the composition, as services are always separable from their composition. It follows that such modelling of the infrastructure only needs to be done to the extent necessary to achieve the desired level of autonomy. For example, in our early example of a book broking service, the business processes themselves are likely to be of much greater significance in measuring service performance than, say, the speed of the network connection used. This incremental approach to infrastructure modelling avoids the need to implement heavy-weight autonomic solutions that require unnecessary pervasive instrumentation and costly monitoring.

## 5. Conclusion

Exogenous management facilitates the recursive construction of autonomic composites that monitor and control the interactions between their constituent services. Architectures that use exogenous management exhibit a number of features including self-management; separability; recursive composition; and grounding through the monitoring of interactions. In this approach, autonomy is a property of the relationships between elements, rather than a property of the elements themselves. Greater levels of autonomy can be incrementally added to a system by replacing basic services with service composites that have a more refined level of modelling, monitoring and control. This incremental approach increases the modularity of the service composition, and helps manage the complexity of the autonomic application.

The ROAD architectural meta-model realises the concept of exogenous management with a few fundamental concepts: *roles* (which are played by services), *contracts* (which are connectors between roles) and *self-managed composites* that have internal *organisers*. The ROAD framework implements this meta-model by providing a set of abstract classes. These classes are extended by the application developer to create autonomic service composites and applications. ROAD contracts are implemented using *association aspects* [4] which intercept and monitor messages as they pass between service roles in a composite. A detailed discussion of the ROAD framework, its capabilities and its performance characteristics can be found in [3].

Further work needs to be undertaken to integrate this approach with mechanisms of service selection and for the negotiation of SLAs between services, both basic and composite. Given that the exogenous nature of the ROAD approach facilitates the composition of systems made up of heterogeneous services, these services could include other types of autonomic *element* that are being developed as part of the current research into autonomic systems.

## 6. References

- [1] Ashby, W. R. *An introduction to cybernetics*, London: Chapman & Hall, 1956.
- [2] Chen, S., Liu, Y., Gorton, I., and Liu, A., "Performance Prediction of Component-based Systems" *Elsevier Journal of Systems and Software*, vol.74(1), 2005, pp. 35-43.
- [3] Colman, A., *Role-Oriented Adaptive Design*. PhD Thesis, Swinburne University of Technology, <http://www.ict.swin.edu.au/personal/acolman/pub/ColmanROADThesis.pdf>, 2006.
- [4] Colman, A. and Han, J., "Using Associations Aspects to Implement Organisational Contracts" *Electronic Notes in Theoretical Computer Science - Proceedings of the 1st International Workshop on Coordination and Organisation (CoOrg 2005)*, vol.150(3), 2006, pp. 37-53.
- [5] Colman, A. and Han, J., "Using Role-based Coordination to Achieve Software Adaptability" *Science of Computer Programming, Elsevier*, vol.64(2), 2007, pp. 223-245.
- [6] Farha, R., Kim, M. S., Leon-Garcia, A., and Hong, J. W. K., "Autonomic Service Architecture using Virtual Services," *Proceedings of Asia & Pacific Network Operations & Management (APNOMS)*, 2005
- [7] Farha, R. and Leon-Garcia, A., "Blueprint for an Autonomic Service Architecture," *Proceedings of the 2006 International Conf. on Autonomic and Autonomous Systems, 2006. ICAS '06*. 2006
- [8] Graupner, S., Kotov, V., Andrzejak, A., and Trinks, H., "Adaptive Control Overlay for Service Management," *Workshop on the Design of Self-Managing Systems, The 2003 International Conf. on Dependable Systems and Networks (DSN-2003)*, 2003
- [9] Kephart, J. O. and Chess, D. M., "The Vision of Autonomic Computing" *Computer, IEEE Computer Society Press Los Alamitos, CA, USA*, vol.36(1), 2003, pp. 41-50.
- [10] Menasce, D. A., "QoS-aware software components" *IEEE Internet Computing*, vol.8(2), 2004, pp. 91-93.
- [11] Mintzberg, H. *Structure in fives: designing effective organizations*, Englewood-Cliffs, New Jersey: Prentice Hall, 1983.
- [12] OASIS "Web Services Distributed Management - Management of Web Services 1.0, OASIS Standard, 9 March 2005" *wsdm-mows-1.0*, 2005
- [13] Sakurai, K., Masuhara, H., Ubayashi, N., Matsuura, S., and Komiyama, S., "Design and Implementation of an Aspect Instantiation Mechanism" *Transactions on Aspect-Oriented Software Development, LNCS*, vol.3880, 2006, pp. 259-292.
- [14] Simon, H. A. *The sciences of the artificial*, Cambridge: M.I.T. Press, 1969.
- [15] White, S. R., Hanson, J. E., Whalley, I., Chess, D. M., and Kephart, J. O., "An Architectural Approach to Autonomic Computing," *First International Conf. on Autonomic Computing (ICAC'04)*, 2004, pp.2-9.