Organisational abstractions for adaptive systems

Alan Colman and Jun Han
{acolman,jhan}@it.swin.edu.au
14 June 2004
## Contents

1. Introduction ................................................................................................................... 3  
   1.1. Structure of this paper ........................................................................................... 3  
2. Conceptual underpinnings ............................................................................................ 4 
   2.1. Adaptation ........................................................................................................... 4 
   2.2. Organisation ........................................................................................................ 6 
   2.3. Component capability and modes of organisations .............................................. 7  
3. An adaptive role-oriented methodology ...................................................................... 8 
   3.1. Decoupling using Design Patterns ...................................................................... 9  
   3.2. Decoupling using roles ...................................................................................... 10  
   3.3. Organisation of decoupled structures based on role associations .................... 10  
   3.4. A meta-model based on the separation of functional and management roles ...... 11  
   3.5. Functional roles ............................................................................................... 12  
   3.6. Operational management roles ......................................................................... 12  
   3.7. Operational-management contracts ................................................................... 14  
   3.8. Organisational management roles ..................................................................... 15  
4. An example .................................................................................................................... 17  
5. Related work ............................................................................................................... 20  
6. Conclusion and further work ..................................................................................... 20  
7. References ................................................................................................................... 22  

*This paper is an extended version of:*

*Colman A and Han, J. 'Organisational Abstractions for Adaptive Systems' submitted to the Hawaiian International Conference on System Sciences 38*
Abstract
Computing environments are becoming more open, distributed and pervasive. The software we build for these dynamic environments will need to become more adaptable and adaptive. This paper introduces a methodology based on ontogenic adaptation — the ability of a system to alter its structure while maintaining its organisational viability. This approach extends existing work on the separation of roles from objects, by defining an organisational layer of abstraction based on the separation of operational-management roles from functional roles. Dynamic role-object bindings and role-role associations are created to form a flexible organisation that can be adapted by an organisational management role. The methodology is illustrated with an example to contrast it with a traditional object-oriented approach.

1. Introduction
As modern computing environments become more open, distributed and pervasive, the software we build for those dynamic environments will need to become more adaptable and adaptive. In this paper we will explore what abstractions make software amenable to adaptation. In a changing environment, where goals of a system may also change, adaptable software needs to achieve its dynamic goals while maintaining its viability. We will argue that explicit organisational abstractions are necessary for designing and building systems in complex, open environments. We base these abstractions on the concept of ontogenic adaptation. We introduce an object-role-oriented software development methodology that uses organisational abstractions.

1.1. Structure of this paper
Section 2 defines the concepts on which our discussion of adaptation and organisation in software is based. We take the concept of ontogenic adaptation from biology and apply it to software systems. The principles of component interchangeability and structural plasticity (flexibility) that are the basis of ontogenic adaptation are prerequisites for maintaining organisational viability in dynamic environments. The nature of organisation is then discussed and we note the differences between emergent and goal-directed organisation. Organisational descriptions of goal-directed software systems need to include a representation of the transmission of goals (control) through the system. Section 2 concludes with a schema of the relationship between the type of software component and the type of software organisation. Organisation varies depending on the capability and autonomy of the components. In these terms, components range from simple objects to autonomous deliberative agents. Within this range, we limit our discussion to objects in adaptive role-based organisations.

Section 3 defines a method of achieving ontogenic adaptation in object-oriented systems. This method achieves structural plasticity by decoupling the relationships between objects and the roles they play, and decoupling the relationships between roles by using explicit associations. We define three types of role — functional roles, operational-management roles and organisational-management roles. Functional roles are elaborated in decoupled class-role model. An abstract organisational structure that represents the control regime and topology of the software system is then developed from the operational management roles. These operational management roles are then bound to functional roles. The organisational structure is then instantiated by binding roles to the objects that play those roles. This plastic structure is adaptable. Once this flexible structure is in place, run-time adaptivity is achieved through the definition of organisational-management roles that create and destroy the object-role bindings and the role-role associations. Program implementation mechanisms for the treatment of objects, roles and organisation as separate concerns are briefly discussed. Section 4 applies our methodology to an example, illustrating the methodology’s differences with a traditional object-oriented approach. Section 5 discusses related work and Section 6 draws conclusions and examines work to be done.
2. Conceptual underpinnings

In this section, we propose a framework for defining adaptation and organisation and their interdependence. Software organisation is then broadly characterised in terms of the capability of the constituent components. Within the range of modes of organisation we identify, the focus in this paper is on adaptive role-based organisation.

2.1. Adaptation

Adaptation is a relationship between a system and its environment. Systems are often classified as adaptable (able to be modified by an external agent) and/or adaptive (able to change itself). However, this distinction does not illuminate a number of important aspects of adaptation. These include:

- What aspects/qualities/parts in the system are subject to change and what aspects remain invariant?
- What motivates the need for change – is it change of goals or requirements of the system, or is it environmental perturbation?
- What are the limits to adaptation? Every system, living or designed, exists in an environmental context and all adaptation is limited. Systems are not in themselves adaptable – they are adaptable with respect to a set of environmental states. Even systems that we regard as highly adaptive (such as humans) are only viable within a limited range of environmental conditions (atmospheric composition, temperature etc.) and within specific ecology.
- To what extent can a system cope with unanticipated changes to the environment or its goals?
- Can the system change the environment? In designed software systems we draw a boundary between the system and the environment, and tend to assume the environment cannot be changed. Adaptation, however, expresses a relationship between a system and its environment. For example, it is humans’ ability to modify their environment that has made them so adaptable.

The question is not whether or not a software system is adaptable, but to what type of change is it adaptable and to what degree of change can the system adapt. We need to determine how much capability for adaptation a system should have, and to what extent can/should we change the system’s environment.

In order to better build adaptable/adaptive systems we need a more refined understanding of the nature of adaptation. Biological systems have been a source of inspiration for the development of adaptable/adaptive software systems. In biological systems two mechanisms of adaptation are commonly characterised (Maturana and Varela, 1980) — evolutionary (phylogenetic) and ontogenic (or ontogenetic) adaptation. If we define adaptation as compatibility between a system and its environment we can add a third category — environmental manipulation. Each of these mechanisms has parallels in designed systems. These three mechanisms are based respectively on reproduction, self-production and production.

Evolutionary (phylogenetic) adaptation is a selective mechanism whereby instances of a class of system reproduce themselves with variations. The variants that are better adapted are selected. Evolutionary adaptation is adaptation of a species (phylogeny). In biological systems this variation is random (“blind variation”) and environmental conditions determine selection. In software systems, evolutionary adaptation has analogies at both design-time and runtime. At design time the versions of a product throughout the software development lifecycle could be regarded as variations. Unlike biological systems, these variations are not “blind”. Versions of the software product are based on models the designers hold and evolve in the iterative development cycle. Versions are tested to ensure they meet requirements (i.e.
are well adapted to the nominal environment). Genetic algorithms are an example of this type of adaptation at runtime. Evolutionary adaptation is based on reproduction of the system. A software system that is amenable to this type of adaptability is “modifiable” or “adaptable”.

Ontogeny is the history of structural transformations of an individual system. Ontogenic (or ontogenetic) adaptation is the ability of a system to change its structure as it interacts with the environment. This change of structure is of two types. The first is the change or interchange of the elements within the structure. In biological systems an example of this type of change is the death and replacement of cells. In software or hardware systems an analogy would be the replacement of one component by another component with a compatible interface. The second type of change is change in the relationships between the elements that make up the system. In animals this plasticity of structure is achieved by the nervous system. The nervous system (including the brain) modulates the interactions between various components of the living system by continually modifying itself. This malleability of the nervous system (the individual self-structuring to fit the environment) is the basis of cognition, learning and social behaviour (Maturana and Varela, 1987). Ontogenic adaptation is based on self-production (autopoiesis) (Maturana and Varela, 1980) — the ability of the system to maintain its organisational integrity even through the elements within it may change and the relationships between those elements change. This alteration of the plastic structure occurs even through the components of the system are structurally-coupled to the environment. In biological systems, the plasticity of structure that enables ontogenic adaptation arises through evolutionary adaptation. In software systems we must design this plasticity into the system. In designed systems a prerequisite of ontogenic adaptation the loose coupling of the system elements that permits the alteration of structure. This alteration of structure can occur at design-time or runtime. At design time the plasticity reduces the costs of making changes to the system. It makes it more adaptable. At runtime, some mechanism for learning is necessary in order that better configurations can be found. This learning could be representational or non-representational or both. Neural networks are an example of non-representational ontogenic adaptation. A software system that has this ability at runtime would be considered adaptive.

There is a third type of adaptation – ability to change the environmental constraints to suit the system. This type of adaptation is also apparent in biological systems. Biological examples include symbiosis, ant pheromone trails to food sources, the creation of nests to moderate environmental perturbations, agriculture and culture. Adaptation through environmental manipulation can be thought of as production. In software engineering the co-evolution of a system and its environment has long been recognised. The design of software systems has been characterised as interactions between “system designers” and “environmental designers” (Sykes, 2003). The boundary between system and environment is typically arbitrarily defined by the constraints of the software development business. Whether or not the environment can be changed is often a negotiated socio-technical decision. In run-time systems depending on where we draw the system boundary middleware systems could be considered environmental. Internet worms and denial of service attacks serve as negative examples of environmental manipulation. However, as software systems are developed for more complex and open software environments, the possibility of environmental constraint manipulation becomes feasible. An example of such environmental alteration is the emergence of social norms in multi-agent societies (Zambonelli et al. 2000).

Adaptation through evolution, ontogenesis, and environmental manipulation are complementary. In this paper, we propose a form of software adaptation analogous to ontogenic adaptation in biology. In biological systems, the plasticity of structure that enables ontogenic adaptation arises through evolutionary adaptation. In software systems we must design this plasticity into the system. In summary:

Ontogenic adaptation = component interchange + structural plasticity + organisational regulation
We will set out a methodology for conceiving and building software that can be adapted, at
design or runtime, to maintain its organisational viability. To do this we will need to define
what organisation is, how it might be represented, and how the organisational structure might
be manipulated to achieve adaptation.

2.2. Organisation

A description of a system’s organisation is a description of the relationships between elements
in that system. Organisation is an abstraction over associations. There are many definitions
of organisation. Parunak and Brueckner (Parunak and Brueckner, 2003) suggest three
complementary aspects of organisation based on information entropy, process, and (emergent)
structure. Organisation1 (O1) is inverse to the amount of entropy or symmetry in the system
based on spatial, functional or temporal regularity; O2 is as process in which O1 increases in
time; and O3 is structure resulting from O2 which can be measured with O1. This definition
nicely binds the physical and informational aspects of organisation together but addresses
only emergent structures.

For software systems that are designed to achieve goals the above definition needs to be
modified to take account of the organisation’s purpose. The process (O2) can include activities
to deliberately modify the structure (O3) to achieve the systems purpose. Organisational
descriptions are means-end functional descriptions and are at a higher level abstraction than
either process or state perspectives. The function may be designed to achieve a purpose
(teleological) or the function may be emergent and have its function ascribed to it by an
observer (teleonomical). Biological systems are not teleological – they only have to maintain
their identity with environmental perturbations. Software systems are teleological – they are
designed for a purpose. An adaptive software system may have to deal with changing goals as
well as changing environments. A complete organisational description of a software system
includes representation of purpose/function and how that purpose is achieved through a
control regime, rather just structure or interaction. Such a description needs to address two
types of goals — how the system achieves first-order goals (functional requirements that tell
us why the system was built) and also how the system achieves higher-order goals such as
system and environment maintenance. Such maintenance of organisational integrity (liveness
and safety) is primary concern in ontogenic adaptation.

The shortcoming of many organisational descriptions is that they reduce the description of
organisation to just the topological structure or to just the process. The perspective of process
and state are partial perspectives. A complete organisational description would need to
indicate how goals are transmitted through the system, how the system changes in response to
changing goals and environmental perturbations, and how it maintains its organisational
integrity. To be useful as an input to the design process, such descriptions need to be based
on principles of organisation that are more than descriptions as particular interactions in a
domain specific system. Certain architectural styles (Shaw and Garlan, 1996) and system
patterns (Bass, Clements, and Kazman, 1998) are examples of the expression of such
principles in domain independent form. This paper will demonstrate organisational
descriptions can be based on the conceptual separation of control from process— the
separation of management of the process from the process itself. Management functions can
be characterised in a domain independent way. These functions include coordination, goal-
transmission, regulation, resource allocation, audit and reporting. The organisational
perspective is one be seen as one a number of possible perspectives, but a perspective that
allows us to explicitly represent and incorporate adaptive mechanisms into a software system.
2.3. **Component capability and modes of organisations**

In order to discuss organisation in software systems, we need to characterise the nature of the software elements being organised. In particular, we need to characterise the degree of capability and the degree of autonomy of those software components. The capability of software components can range from passive objects that respond deterministically to a set range of inputs, to intelligent agents that actively sense their environment and deliberate on courses of action. In this schema there is no sharp distinction between components and agents. Agents are components that exhibit greater degrees of autonomy. Autonomy of components can vary from objects that respond deterministically to requests, to components or agents with varying degrees of autonomy — autonomy to take action, to set goals and even to violate certain types of constraint.

The type of component combined with the different types of control or constraint that restrict component autonomy result in different modes of organisation. In order of increasing autonomy of the components/agents these modes of organisation can be characterised as:

- **Command based organisation** - an organisation formed as a command hierarchy would have strong central goals while the components having little autonomy.
- **Fixed role-based organisation** - an organisation based on roles would maintain central goals, but components or agents would have autonomy of action to carry their roles.
- **Adaptive role-based organisation** - an organisation where there is a loose coupling between components and roles. Roles may be performed by different components and components may have the autonomy to adopt roles. Relationships between roles, and thus components, can also change.
- **Emergent cooperative organisation** – a protocol-based organisation gives the agent goal and action autonomy within certain constraints (norms, laws, access to resources etc). Agents share social goals and act cooperatively within the constraints to achieve these goals. These constraints would be aimed at maintaining the viability of the system and at the optimisation of utility.
- **Laissez-faire** - At the extreme, there may be no explicit social goals or organisation. In such a libertarian society, agents would be able to act and set goals as they please. Structure would still emerge but be based on individual agent’s capability, power and ability to access resources – the ‘law of the jungle’ prevails.

Listed below is the relationship between degrees of component autonomy and type of social organisation.

**Table 1: Autonomy of software components and types of organisation**

<table>
<thead>
<tr>
<th>Organisation Autonomy of component</th>
<th>Command hierarchy</th>
<th>Fixed roles based organisation</th>
<th>Adaptive roles based organisation</th>
<th>Emergent organisation (cooperative)</th>
<th>No organisation (laissez–faire competitive)</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Action</td>
<td>x</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Goal</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Constraint</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strong organisational goals</th>
<th>Designed structure</th>
<th>Weak organisational goals</th>
<th>Emergent structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less Autonomy</td>
<td>Strong organisational goals</td>
<td>More Autonomy</td>
<td>Weak organisational goals</td>
</tr>
</tbody>
</table>

In any real-world organisation/society, a combination of these modes of organisation are likely to be present. For example, an organisation such as an army that might be primarily characterised as a command-and-control organisation, would also be influenced by role-based and emergent structures. The formal authority structure in an army is likely to be modified by informal networks of communication and influence, established norms of behaviour.
capability of the officer-agents etc. In designed closed computer systems the command-and-control mode of organisation predominates. However, the achievement of goals in such an organisation is qualified by its ability to access resources and so on. As computing moves to a move open architectures, role-based and emergent organisational modes will become more important.

Organisational structure is designed or emerges from component interactions which are mediated by structuring mechanisms. These structuring mechanisms and their relationship to modes of organisation are illustrated below.

Table 2 Modes of Organisation and their Structuring Mechanisms

<table>
<thead>
<tr>
<th>Mode of organisation</th>
<th>Mechanism of structuring</th>
<th>Software Methodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command structure</td>
<td>Designed organisational control</td>
<td>Structured</td>
</tr>
<tr>
<td>Fixed role-based structure</td>
<td>System components have fixed organisational roles</td>
<td>Object-oriented</td>
</tr>
<tr>
<td>Adaptive role-based structure</td>
<td>Loosely structured organisation with dynamic roles. Designed regulatory mechanisms (management and laws)</td>
<td>Role-oriented</td>
</tr>
<tr>
<td>Cooperative structures</td>
<td>Designed regulatory mechanisms (laws) and emergent regulatory mechanisms (norms)</td>
<td>Agent-oriented (Cooperative CDPS)</td>
</tr>
<tr>
<td>Competitive laissez-faire</td>
<td>Environmental structuring (emergent structure only)</td>
<td>Agent-oriented (open systems)</td>
</tr>
</tbody>
</table>

In this paper, we will focus organisations with role-based object-oriented structures that are structured by the assignment of organisational responsibilities.

3. An adaptive role-oriented methodology

In this section we introduce a methodology for creating adaptable and adaptive software based on the organisation of flexible object-role structures. The activities of the methodology are discussed in the following subsections:

1. Object structures are decoupled to make them more flexible by using design patterns and role-object separation.
2. The types of dynamic role-object bindings and role-role associations from which adaptable organisation structures be built are identified.
3. There is a crucial distinction between management roles and functional roles. In our methodology, organisation is an abstraction based on management roles. Two types of management role are identified — operational and organisational.
4. An abstract organisation is defined from operational-management roles. The network of operational-management roles is an expression of the control regime and topology of the organisation.
5. The binding of operational-management roles to functional roles creates a domain specific organisation. When objects are bound to these roles, an instantiated organisation is created.
6. Organisational-management roles are defined to provide control of the organisational structure via the manipulation of role-object bindings and role-role associations.

Traditional object-oriented systems are based on fixed-role organisation. The associations between objects are set in the public methods of those objects and the calls to those methods from other objects. These method invocations between objects are scattered through the code. While objects with fixed roles may be modelled in the design, explicit representation of roles is lost in the implementation. Even though the specific interactions between objects are modelled using sequence and interaction diagrams, the organisation expressed by those
associations is not explicitly modelled in the design. As well as being implicit, the organisational structure can be difficult to change at design time and is largely frozen at compile-time.

As pointed out above, the prerequisites for ontogenic adaptation are interchangeability of components and plasticity of structure. In traditional object-oriented programming, partial adaptability is achieved by separating the interface from the implementation of the object. Object implementations can be swapped within the rigid structure defined by pre-existing objects. Interchange is made possible at design-time through well-defined interfaces that allow class implementations to be interchanged. At runtime dynamic binding of polymorphic objects in an inheritance hierarchy also facilitates object interchange.

However, traditional object-oriented approaches do not address the second prerequisite of ontogenic adaptation - that of plasticity of structure. Plasticity of structure can be achieved by creating levels of indirection between program elements (rather than just indirection between interface and implementation). Over recent years a number approaches have been developed to create systems with more loosely coupled elements thereby allowing a degree of adaptability. These developments of the object-oriented approach include design patterns, and role-based methodologies.

### 3.1 Decoupling using Design Patterns

While design patterns are usually viewed repositories that facilitate design reuse, a number of them can be viewed as mechanisms for decoupling the structure of a program. In GOF (gang-of-four) design patterns (Gamma, Vlissides, Johnson, and Helm, 1995), interactions are programmed to an abstract interface rather than an implementation of the object. The patterns use configurations of objects further enhance the adaptability of the program. These configurations consist, in part, of objects which do not correspond to concepts in the problem domain (e.g. Abstract Factory). Such configurations provide indirection of implementation on how objects are created, how they are structured and how they behave. The indirection provided by these patterns allows a general structure of interaction to be defined independent of the specific implementations of objects that make up the structure. For example, the Adaptor pattern allows indirection between the associations of objects; the Decorator pattern provides indirection between object identity and its behaviour. The Table 3 below, we have categorised some of the types of indirection introduced by various GOF patterns.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Purpose</th>
<th>Indirection between</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptor</td>
<td>Creation</td>
<td>association of objects</td>
</tr>
<tr>
<td>Factory</td>
<td>Creation</td>
<td>object and its composition</td>
</tr>
<tr>
<td>Decorator</td>
<td>Structure</td>
<td>object identity and object behaviour</td>
</tr>
<tr>
<td>Composite</td>
<td>Structure</td>
<td>object reference and its cardinality</td>
</tr>
<tr>
<td>MVC</td>
<td>Structure</td>
<td>objects and the view of those objects</td>
</tr>
<tr>
<td>State</td>
<td>Behavior</td>
<td>object and its state</td>
</tr>
<tr>
<td>Strategy</td>
<td>Behavior</td>
<td>object and its methods</td>
</tr>
<tr>
<td>Visitor</td>
<td>Behavior</td>
<td>data structure and operation on the data</td>
</tr>
</tbody>
</table>

Designing object-oriented systems using patterns, such as those in Table 3, can partially reduce the tight coupling of the structure. GOF patterns are categorised according to their “purpose” depending on whether the relations effect object creation, structure or behaviour. This purpose is a high-level, domain-independent descriptor for what the pattern does at an
organisational level. As such, design pattern descriptions have an organisational aspect, although their scope is usually limited to fragments of design rather than being system-wide.

3.2. Decoupling using roles

The separation of interface from object is also apparent in the notion of a role. A role is an interface of an object that satisfies responsibilities to the system as a whole. Roles can be added to, and removed from, objects.

A number of authors propose that roles should be treated as first class modeling and programming entities (Kendall, 1999b; Kristensen and Osterbye, 1996; Lee and Bae, 2002). Kristensen (Kristensen and Osterbye, 1996) provides a conceptual model for roles. The definition of roles is based on the distinction between intrinsic and extrinsic members (methods and data) of an object. Intrinsic members provide the core functionality of the object, while extrinsic members contain the functionality of the role. In our view, this ‘core functionality’ is the situated computational and communication capabilities of the object. Extrinsic members are at a different level of abstraction— they define the domain function of the object.

Roles can be formed into control and abstraction hierarchies, they can be aggregated and dynamically assigned to objects. We can categorise role model approaches in terms of how much independent identity the roles have. In traditional OO, roles only exist as an intrinsic property of a class and express its external relationships with other objects. Hybrid approaches (Kendall, 1999a) (Kristensen and Osterbye, 1996) encapsulate roles but allows them no existence separate to the objects to which they are bound. At the far end of the spectrum there is a radical separation of object from roles (Lee and Bae, 2002). In this approach, all external behaviour of the object should be encapsulated in roles, and that these roles are first-class entities with their own identity. Issues that arise from role-oriented methodologies include role-role compatibility and object-role compatibility. We will not address these issues here other than to note that Class Dictionary Graphs (Lieberherr, 1996) and Service Model in Gaia (Zambonelli et al. 2003) provide general solutions to compatibility between such design elements.

3.3. Organisation of decoupled structures based on role associations

Organisation in object-oriented systems can be viewed as a network of roles. Organisational descriptions need to express both a topology and a goal-oriented control regime for the whole system (Zambonelli et al. 2003). This role organisation is a ‘global control-flow abstraction’ (Lee and Bae, 2002). Adaptable systems need to be able to change this topology and control regime. Figure 2 below illustrates three level of abstraction in a role-based object-oriented system. Roles are always defined in relation to other roles. These role-pairs form associations. Organisational descriptions are abstractions at the level of role associations.

The diagram illustrates a number of possible types of association. Associations can be between single or aggregate roles (e.g. R₂). An object may adopt a number of roles (e.g. O₂). If there is redundancy in the system, the same role might be performed by different objects.
Objects that have fixed behavior (such as resources) can be part of a role-object association (illustrated by the dash-dot line).

Figure 2 Associations in levels of abstraction in a role-based object-oriented system.

While design patterns and encapsulated roles may provide the prerequisite plasticity of structure, they do not provide global organisational abstractions that will guide the creation of structures that enable the achievement of first-order goals and ensure the system remains viable in changing environments. Some ‘system patterns’ (Bass, Clements, and Kazman, 1998) such as the compound Bureaucracy pattern (Riehle, 1998) and SupplyChain patterns do provide organisational abstractions. These can be considered proper, if primitive, organisational patterns in that they represent the global flow of control in the system. We will examine the Bureaucracy pattern in more detail below. In the next section we will discuss the concept of management roles from which such organisational descriptions can be elaborated.

3.4. A meta-model based on the separation of functional and management roles

In order to create organisational abstractions based on roles, we need to distinguish three types of role. We define these as follows:

- **Functional roles** – these are focused on first-order goals – on achieving the desired problem-domain output or environmental change. Functional roles constitute the process as opposed to the control of the system. Some functional roles are coupled to the environment through system i/o.

- **Operational management roles** – these focus on regulating the process so that the system can achieve its functional goals. This distinction is based on a conceptual separation of management control from process. The network of operational management roles represents the global control flow of the system (as opposed to data flow). Operational management roles ensure that the system copes with to environmental perturbation within predefined limits. Operational management roles have no direct connection with the environment.

- **Organisational management roles** maintain a reflective representation of the system’s organisation and mechanisms for restructing the organisation by creating/destroying role-object bindings and dynamic role-role associations. The controllers (objects/agents/humans) that play organisational management roles are responsible for the restructuring of the (sub)system defined by the network of operational-management roles. These controllers are linked to the environment and monitor the performance of the software system in terms of its goals. This forms an adaptive loop.
An overview of the relationship between the three types of roles in a system and the environment is shown in Figure 3 below.

3.5. Functional roles

Object-oriented classes are commonly modeled on the basis of problem domain responsibilities. These domain responsibilities are captured in functional roles. The separation of extrinsic behavior in functional-roles from intrinsic behavior is discussed in Section 3.2 above. However, few object-oriented programs consist of only functional objects/roles. It is a common practice to create implementation-specific management classes that perform object creation, aggregation and coordination functions. Many such configurations are documented in the design patterns discussed above. However, these individual management classes or configurations are not placed in a global organisational framework. We do this through a network of operational-management roles.

3.6. Operational management roles

Operational management is responsible for maintaining control flow though an existing organisational structure. In particular an operational manager would:

- Allocate tasks to objects performing subordinate roles
- Receive commands from supervisors and send commands to subordinates
- Regulate the process
- Balance load amongst subordinate objects
- Report the state of the process
- Allocate available resources to subordinate objects performing roles

An example an arrangement of operational management roles is the Bureaucracy pattern (Riehle, 1998). The Bureaucracy pattern is a compound pattern (or “system pattern” (Bass, Clements, and Kazman, 1998)) formed from the Mediator, Chain of Responsibility, Composite and Observer GOF patterns (Gamma, Vlissides, Johnson, and Helm, 1995). As well as their functional roles, objects can take on management roles within the organisational
hierarchy of the system. The management-type roles within the bureaucratic hierarchy include Clerk, Supervisor (in (Riehle, 1998) called a “Manager”), Subordinate and Director. Every object within the hierarchy plays the roles of a Clerk and also either a Supervisor or Subordinate or both. A Director is a Supervisor who itself has no Supervisor, and as such represents the root of the hierarchy. These roles are themselves composite roles from other patterns. For example the Supervisor plays the Mediator, Successor and Observer roles from the constituent patterns. The external interaction with the Bureaucracy is captured in the ClerkClient and DirectorClient roles which interact, respectively, with Clerk and Director roles. The role diagram for the Bureaucracy pattern is illustrated Figure 4 below.

![Bureaucracy Pattern Diagram](image)

**Figure 4 Bureaucracy Pattern - example of an organisational pattern based on operational management role-classes**

The management type roles listed above (Director, Supervisor, Subordinate etc.) are only one example of a group of roles that can form organisation abstractions. Supply chain roles (SC Successor, SC Predecessor) can also be used to build object structures with a representation of global flow of control in work-flows (Kendall, 1999b). At courser levels of granularity, relationships such as master-slave, client-server and peer-peer can be viewed as expressing organisational relationships. These “architectural styles” are abstract expressions of the topology of components that encapsulated management responsibilities and the run-time control and/or data transfer between these components (Bass, Clements, and Kazman, 1998). According to Garlan (Garlan D., 2000), architecture should express “separation of concerns about the functionality of a component from the ways in which a component is connected to (interacts with) other components”. Unfortunately, while these styles provide a way of systematising and codifying high-level interactional patterns, Architectural Description Languages typically reduce the description to a static view of components and connectors in which the organisational perspective is lost.

The separation of operational management roles from functional roles gives us a way to describe the organisational structure of the system and the control regime of that structure. A definition of an abstract organisation based on the bureaucracy pattern is show below (objects attached to roles are not shown for simplicity). Note that the Clerk role is not shown because it is an abstract role — an object cannot adopt a Clerk role without adopting a Director, Supervisor or Subordinate role.
Hierarchies of any complexity and topology can be created from an organisational pattern such as Bureaucracy. A domain specific elaboration of such a hierarchy would add domain-specific responsibilities / functions / roles to a position in the hierarchy. The static representation of such a hierarchy would have the appearance of a business’s organisational chart. A number of categories of topologies have been suggested in (Zambonelli et al. 2003) for agent organisations. These could also be applied to object-oriented organisations:

- Collection of peers – small organisations where communication costs are low and communication and collective decision making possible between all members
- Hierarchy – supervisors assumes responsibility of coordination activities
- Multilevel hierarchy – a hierarchy of supervisors
- Complex Topology – some combination of the above

### 3.7. Operational-management contracts

Figure 5 is an abstract organisation because it only shows management roles. These management roles are not yet attached to functional roles or the objects that play those roles. For example, if the organisation was instantiated as a retail business, the object that plays the highlighted Subordinate role (in Figure 5) might also play the functional role of SalesPerson. Figure 10 in the next section gives an example of an instantiated organisation.

Because every functional role would have a position in the organisational structure, it must have one or more associated operational-management roles — one for every type of role-role association in which the role participates. Because every operational-management role association would also represent a link in the organisational structure, there is a correspondence between functional role-role associations and operational role-role association. Such role-role associations can be viewed as contracts that they restrict the interactions between objects playing roles. In traditional object-oriented programming, an object will respond to any valid invocations of its public methods. Operational-management role-role contracts restrict the type of method one role can invoke in another role, or what methods it will respond to from another role.

Operational-management contracts can be defined in terms of communicative act (CA) primitives (The Foundation for Physical Intelligent Agents, 2002) — the types of message it is permissible for one role to send its associated role and the expected response. For example, if such primitives are defined as Invoke, Inform, Query, Accept, Refuse, ResourceAllocate and ResourceRequest, then a Supervisor-Subordinate contract of permissible interactions might be defined as follows:
Table 4 Example operational-management contract

<table>
<thead>
<tr>
<th>Operational-management Contract</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Party A</td>
</tr>
<tr>
<td>Party B</td>
</tr>
<tr>
<td>A initiated</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>B initiated</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

When the functional role in an organisational structure is bound to the operational-management role using such a contact, all functional role invocations and responses are associated with CA primitives. We will refer to this correspondence between functional-role associations and operational-role associations as association inheritance because of the conceptual similarity to a class implementing an abstract interface. The relationship between objects, functional roles, and operational-management roles is illustrated in Figure 6 below.

Using the example above, suppose two functional roles in a retail business are SalesPerson and AreaManager, and the corresponding operational roles are Subordinate and Supervisor. The operational-management role association would restrict interactions between the object playing the SalesPerson and the AreaManager to certain types of interaction. For example, the method SalesPerson.setSalesTarget( ) can only be invoked by an AreaManager. When the roles are instantiated by binding to objects, it is only the particular object instance(s) playing the role of AreaManager that can invoke the setSalesTarget method associated with the object playing the role SalesPerson. Once functional roles are bound to operational-management roles we have a domain-specific representation of the organisation.

A number of types of operational-management roles can be identified. These include:

- Supervisor-subordinate
- Auditor-auditee
- Peer-peer
- Supply-chain predecessor-successor
- Production-line predecessor-successor

Further work needs to be undertaking to determine how these can be defined on a more fundamental level.

3.8. Organisational management roles

To be ontogenically adaptive, an organisation needs to include roles that are concerned with controlling and modifying its organisational structure rather than just managing the process or flow of control through the organisation. In an adaptable organisation this role would be
performed at design-time. Encapsulating such roles should make the system more maintainable. This form of higher-level management we will call organizational management because what is being managed is the structure of the organisation itself. These roles would be responsible for creating and maintaining an organisational structure, such as illustrated in Figure 5, by creating and destroying object-role bindings and role-role association links. Organisational management responsibilities include:

- Allocation of functional role or operational-management to an object (instantiation of role through role-object binding).
- De-allocation of role from an object.
- Creating / destroying dynamic association instances between pairs of roles.
- Aggregation / disaggregation of roles. This could include binding of functional role to an operational-management role.
- Checking for organisational integrity – for example, checking for uninstantiated roles or incompatible bindings.
- Monitoring system performance in the environment.
- Global resource management. In more open systems, this might also include service discovery.

These higher level management functions include role allocation to objects, role creation, merging and splitting. Different regimes could be used for determining roles allocation and reallocation. A number of these have been suggested in (Zambonelli et al. 2003). They include:

- Workload partitioning – suitable for division of similar tasks
- Workload specialisation – suitable for differentiated tasks or tasks require particular capabilities or access to resources
- Market models – suitable in very large organisations where competitive and self-interested behaviour are likely to emerge.

Organisational management is separated into a role and a controller. The role maintains a representation of the organisational structure and mechanisms for manipulating the structure. The controller interacts with the interface provided by the role and the environment (see Figure 3 above). The controller could be an object, agent, human operator or designer. It/he/she needs to have some understanding of viable organisation (integrity rules and liveness/safety goals), and how to change the structure in order to achieve effective organisation. This separation has the advantage of making the controller ‘swappable’ and thus enabling evolutionary development of the capability of the controller. As modeling of the adaptive loop improves controllers can be gradually upgraded. In mixed-initiative systems, human controllers can replace machine controllers when the environmental perturbation exceeds the parameters to which the artificial controller can adapt. The more the system has to adapt to unanticipated variation, the more intelligence is required of the controller.

To summarize, role-oriented organisation can be achieved through stages involving delayed binding under the control of a manager playing an organisational-manager role. These stages are:

1. Creating contracts between operational-management roles to form a network. This abstract organisational structure represents the topology of the organisation and global control flow based on permissible role interactions.
2. The binding of functional roles to the operational-management roles. This binding creates a domain-specific organisational structure.
3. The binding of those roles to objects. This creates on instantiation of the organisational structure.
4. An example

In this section we illustrate the application of our methodology with an example. In a business organisation employees perform various roles, and may take on more than one role. A role of an employee in an organisation might be ‘supervisor’ - supervisors can send commands to their subordinates. But supervisors themselves perform the role of employees and may themselves be subordinates. The organisation is defined by the role relationships rather than by the individual who perform the roles. This organisation allows top level goals to be transmitted down through the organisation.

Below we will model a highly simplified business department that makes Widgets and employs Employees with different skills to make them. A traditional class diagram for the entities in this business might look something like the Figure 7 below. Roles (in italics) are implicit in the functionality of the class. Associations between Employee subclasses have been omitted for simplicity. Supervisor and Subordinate could be implemented as abstract interfaces to the Employee class.

![Figure 7 Traditional Object-oriented Class Model](image_url)

In a compositional role-object approach the classes ThingyMaker, DooverMaker, and so on, could be modelled as roles attached to an object of class Employee rather than as a specialisation of Employee (through inheritance). Supervisor and Subordinate are be abstract operational-management roles — they can only be added to concrete associations (e.g. to a Foreman-DooverMaker association).

The decoupled class model and role model of our department is illustrated in Figure 8 below. The Class model is simpler because the static associations with Employee object have been removed. These are replaced by potential role-object bindings. The dynamic associations that link role to each other are also omitted because these bindings are delayed. Note that Widgets are not treated as a role of Products but have a static association. This is because in the problem domain Products cannot change roles.
Figure 8 Partially Decoupled Class and Role Model

The topology of an organisational network based on Figure 8 is illustrated in Figure 9. This organisational structure is a cross-cutting aspect to the class structure. Note the functional role associations inherit restrictions on interaction from the organisational management role associations. The organisational structure is abstract because as yet there are not object-role bindings.

Figure 9 Abstract organisational structure

By binding objects to the roles in the abstract organisation we obtain a substantiated organisational structure. This is illustrated in Figure 10 below.
When the system needs to change its organisation to meet radically changing goals or to adjust to changing environmental/resource constraints (e.g. change in network loads or node availability), this change can be effected by the re-allocation of roles and associations. This is the task of the organisational-management role which monitors the viability and performance of the organisation in the adaptive loop. For example, suppose the computational node on which $e_5$ (in the figure above) runs becomes unavailable due to network failure. Employee $e_1$, who plays an organisational-manager role, might transfer the role of assembler from $e_5$ to $e_4$.

Similarly, if a goal (non-functional requirement) of speed of production changes causing a bottle-neck because of the limited capacity of the $e_3$ object to carry out the Assembler role, additional Assembler positions could be added to the organisational structure.

Organisational management can be either centralized or distributed. If it is distributed to organisational managers, then their authority to alter the structure of the organisation would have a limited scope. The role of an organisation manager WidgetDept-OrganisationalManager has been delegated to the object $e_1$ playing the role of ProductionManager (alternately it could be delegated to the ProductionManager role itself). This, for example, would give that object the authority to alter the organisational relations (object-role bindings and dynamic associations) in the diagram, not including its own role binding ($e_1$ cannot make itself a ProductionManager). For simplicity, the control links between the WidgetDeptOrganisationalManager and each of the object-role bindings have been omitted from the diagram. In the larger organisation, distributing the roles of organisation management across the system creates a degree of self-organisation in the sub-systems or components.
5. Related work

Our methodology extends work on role and associative modeling in (Bäumer et al. 2000; Kendall, 1999a; Kendall, 1999b; Kristensen and Osterbye, 1996; Lee and Bae, 2002). However, these role-oriented approaches do not define an organisational level of abstraction. Our method abstracts an organisational level through the separation functional and management roles.

Structural decoupling is also addressed in design patterns (Bäumer et al. 2000; Gamma, Vlissides, Johnson, and Helm, 1995; Riehle, 1998), although these patterns only describe design fragments. Another, more comprehensive, approach to decoupling is the Demeter method (Lieberherr, 1996; Lieberherr et al. 2001). This method attempts to achieve adaptivity by decoupling the details of a data structure from the operations on those structures. Executable programs are customized from high-level ‘adaptive’ programs. These generic programs are linked to detailed data structures via ‘class dictionary graphs’ rather than patterns of objects. However, the Demeter method is not organisational in that it has a method-focus rather than modeling software as a network of roles. Organisational level abstractions are partially addressed in work on software architecture (Bass, Clements, and Kazman, 1998; Shaw and Garlan, 1996). According to Garlan (Garlan D., 2000), architecture should express “separation of concerns about the functionality of a component from the ways in which a component is connected to (interacts with) other components”. In this sense, as well as representing topology, architectural styles are abstract expressions of the management responsibilities and the run-time control of components (Bass, Clements, and Kazman, 1998). However, as with the above role-oriented approaches, there is no basis for encapsulating the management roles that represent a control regime separately from the functional roles. Architecture is reduced to a static representation of components and connectors which is not adaptable.

Work on roles has also been undertaken in multi-agent software engineering (Juan et al. 2002; Odell et al. 2003; Zambonelli et al. 2003). In particular, (Zambonelli et al. 2003) extends the concept of a role model to an organisational model. MAS systems however rely on components with deliberative capability and more autonomy than the components discussed here. These agents negotiate interactions with other agents to achieve system level goals. These negotiations occur within a more amorphous structure than is defined here.

Like our approach, control-theoretic architectures separate control from functional processes (Shaw, 1995). Such systems are designed to maintain system viability during anticipated environmental perturbation, but they cannot be considered adaptive. Recent work on intelligent control (Herring and Kaplan, 2000) adds an adaptive loop on top of the operational control loop. This is similar to our concept of organisational-management roles that control the structure of the organisation through manipulating the operational-management associations. In control-theoretic approaches the component structure of the system is arbitrary. There is no concept of role or object-role binding. Such systems are not ontogenically adaptive as their structure is fixed.

6. Conclusion and further work

This paper introduces an object-oriented role-based methodology based on the concept of ontogenic adaptation. Ontogenic adaptation requires interchangeable elements, plastic structure and organisational regulation.

In software systems these prerequisites can be achieved through creating decoupled object-role structures. The roles in this initial structure are functional roles. A cross-cutting management role-structure is then created from operational-management roles. Linking these management roles forms an abstract organisational structure that represents the topology of
the organisation and global control flow based on permissible role interactions. The functional roles are then bound to the network of operational-management roles through associative inheritance. This binding creates a domain-specific organisational structure. An instantiated organisational structure can then be created by binding functional roles to objects. Such a structure is ontogenically adaptable. Run-time adaptivity could be achieved by defining organisational-management roles. These roles control and maintain the organisation by creating and destroying object-role bindings and dynamic role-role associations.

A number of possible mechanisms have been suggested for adding roles to objects including aspects (Kendall, 1999a), mixins and the decorator pattern (Kristensen and Osterbye, 1996). In (Kendall, 1999a), various options for implementing these mechanisms are discussed. Kendall (Kendall, 1999b) has shown how aspect-oriented approaches can be used to introduce role behavior to objects. Roles are encapsulated in aspects that are woven into the class structure. Further work needs to be done on appropriate mechanisms for encapsulating functional, operational and organisational management roles.

Different types of role will require differing mechanisms for implementation. If functional roles can swap objects, they will need to be entities with some form of runtime identity. If functional-roles are woven into objects using aspects, then organisational management changes can only occur at compile time. The program would therefore be adaptable rather than adaptive. Operational-management roles, on the other hand, may suitably implemented as aspects (Kendall, 1999a), as the contracts they define are relatively stable. The permissions defined in those contracts crosscut the functional role at various join points (e.g. method invocation). Finally, the domain of organisational-management roles is the organisation itself rather than the problem domain. Some explicit representation of roles and their binding to objects is necessary. Although organisational roles are not in themselves a cross-cutting concern, the monitoring of the organisation may be instrumented using aspects or some other reflective mechanism.

Role-oriented methodologies have a number of outstanding issues including ensuring role-role compatibility and object-role compatibility. Constructs such as Class Dictionary Graphs from the Demeter method (Lieberherr, 1996) and Service Model from Gaia (Zambonelli et al. 2003) may provide general solutions to compatibility between such objects and roles.

The nature of operational management needs to be further developed. Links other than supervisor-subordinate need to be examined and protocols developed for all such associations. It may be possible to develop a pattern language of such operational management associations. Kristensen (Kristensen, 2002) suggests associations could be formed into generalization and aggregation hierarchies.

Finally, object and role based approaches do not in themselves provide models of the environment within the organisational structure which will allow the maintenance of organisational stability. Control-theoretic approaches (Shaw, 1995; Herring and Kaplan, 2000) to building organisational controllers may suggest useful approaches for adding the necessary dynamic environmental coupling.
7. References


