

Organizational abstractions for adaptive systems

Alan Colman and Jun Han
School of Information Technology
Swinburne University of Technology
Melbourne, Victoria, Australia
{acolman,jhan}@it.swin.edu.au

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 developing adaptive systems based on the concept of ontogenic adaptation – the ability of a system to alter its structure while maintaining its organizational viability. This approach extends existing work on the separation of roles from objects, by defining an organizational 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 organization that can be adapted by an organizational 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 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 organizational 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 organizational abstractions.

1.1 Motivation

To highlight the need for adaptive systems and methodologies for building them, we will model a highly simplified business department that makes Widgets and employs Employees with different skills to make them. In such a business organization an employee can perform a number of varied roles — sometimes simultaneously. A traditional object-oriented class diagram for the entities in this business might look something like the Figure 1 below. Roles (in italics) are implicit in the functional

associations between the classes. The associations between Employee subclasses have been omitted for simplicity. Supervisor and Subordinate could be implemented as abstract interfaces to the Employee class.

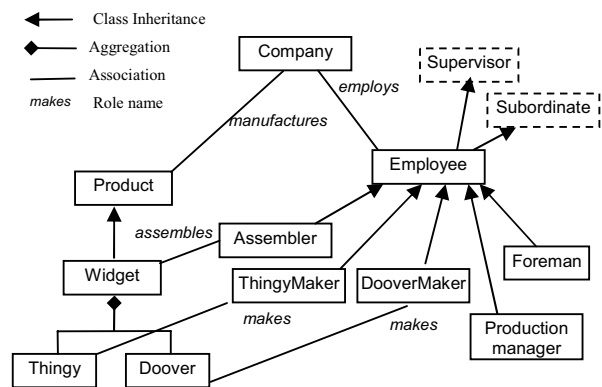


Figure 1 Traditional Object-oriented Class Model

The above traditional class model is static. The associations between classes are fixed at design-time in method invocations and inheritance relationships. The associations between classes/objects cannot be dynamically created or richly described. For example, what *types* of interaction are permissible between an Assembler and a Foreman, and do these interactions differ from those types of interaction between an Assembler and a ThingyMaker? Can an Assembler tell a Foreman what to do by invoking its methods? In object-oriented design, there is no organizational level description in terms of the control of the system. The global flow of control through the structure cannot be represented. Only particular sequences of specific interactions can be shown (e.g. in sequence diagrams). Finally, in traditional object-oriented design, roles are implicit to objects that play them. There can be no dynamic adaptation of the structure of the relationships between objects and roles in response to changing demands on the system.

In this paper we will show how such a structure can be made more adaptable through our methodology that defines of three types of role.

1.2. Structure of this paper

Section 2 defines the concepts on which our discussion of adaptation and organization 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 organizational viability in dynamic environments. The nature of organization is then discussed and we note the differences between emergent and goal-directed organization. Organizational descriptions of goal-directed software systems need to include a representation of the transmission of goals (control) through the system. How this global-control is achieved will vary 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 organizations.

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 *organizational-management* roles. Functional roles are elaborated in decoupled class-role model. An abstract organizational 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 organizational structure is then instantiated by binding roles to the objects that play those roles. Once this flexible structure is in place, run-time *adaptivity* is achieved through the definition of organizational-management roles that create and destroy the object-role bindings and the role-role associations.

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, organization and their interdependence. Software organization is then broadly characterized in terms of the capability of a system's components. Within the range of modes of organization we identify, this paper focuses on 'adaptive role-based organization'.

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 its 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

In order to better build adaptable/adaptive software 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 characterized [15] — evolutionary (phylogenetic) and ontogenic (or ontogenetic) adaptation. In this paper we will focus on ontogenic adaptation of software systems.

Ontogenic adaptation is the ability of an individual system to change its structure as it interacts with the environment. In contrast, *evolutionary* adaptation is the selection of better adapted individuals in a species (phylogeny) through *reproduction*. The self-structuring of the nervous system and the brain in an animal as it develops is an example of ontogenic adaptation. Ontogenic adaptation is based on *self-production* (autopoiesis) [15] — the ability of the system to maintain its **organizational viability** while changing its structure. 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 multi-cellular organisms. An analogy in software or hardware systems would be the replacement of one component by another component with a compatible interface. The second type of change is the modification in the relationships between the elements that make up the system. In animals this plasticity of structure is achieved by the nervous system. In software systems we must design this plasticity into the system. In summary:

Ontogenic adaptation = component interchange + structural plasticity + organizational regulation

We will set out a methodology for conceiving and building software that can be adapted, at design or runtime, to maintain its organizational viability. To do this we will need to define what *organization* is, how it might be represented, and how the organizational structure might be manipulated to achieve adaptation.

2.2. Organization

Organization can be viewed as a *measure of entropy* (the amount of spatial, functional or temporal regularity in a system), a *process* (how that regularity is achieved) or a *structure* (that is the result of such a process) [17]. Unlike natural systems, where organization is emergent, software systems are designed to achieve goals. The organizational process can include activities that alter the organizational structure to achieve the system's purpose. All viable systems must maintain their identity in the midst of environmental perturbations. As software systems are designed to achieve goals, an adaptive software system may also have to deal with changing goals as well as changing environments.

The shortcoming of many organizational descriptions is that they reduce the description of organization to just the topological structure or to just the process. The perspectives of state and process are partial. In our definition, organizational descriptions of designed systems are means-end functional descriptions. They are at a higher level abstraction than either process or state perspectives. A complete organizational 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 the system maintains its organizational viability. To be useful as an input to the design process, such descriptions need to be based on principles of organization that are more than descriptions as particular interactions in a domain specific system. Certain architectural styles [21] and system patterns [1] are examples of the expression of such principles in domain independent form. This paper will demonstrate that organizational 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 characterized in a domain-independent way. These functions include coordination, goal-transmission, regulation, resource-allocation, auditing and reporting. The organizational perspective is one of a number of possible perspectives, but it is a perspective that allows us to explicitly represent and incorporate adaptive mechanisms into a system.

The way a software system is organized will depend on the type of elements that constitute the system. Software components can range from passive functions or objects that respond to a set range of inputs, to intelligent agents that actively sense their environment and deliberate on courses of action. Modes of organization include command structures, role-based organisations and open

protocol-based systems. In many systems, a combination of these modes of organization will be present. In computer systems the command mode of organization predominates. As computing moves to more open architectures, role-based and emergent organizational modes will become more important. In this paper we will focus on adaptive role-based object-oriented organizations that are structured by the assignment of organizational responsibilities.

3. An adaptive role-oriented methodology

In this section we introduce a methodology for creating adaptable and adaptive software based on the organization 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 organization structures be built are identified.
3. There is a crucial distinction between management roles and functional roles. In our methodology, organization is an abstraction based on management roles. Two types of management role are identified — *operational* and *organizational*.
4. An abstract organization is defined from operational-management roles. The network of operational-management roles is an expression of the control regime and topology of the organization.
5. The binding of *operational-management* roles to *functional* roles creates a domain specific organization. When objects are bound to these roles, an instantiated organization is created.
6. Organizational-management roles are defined to provide control of the organizational structure via the manipulation of role-object bindings and role-role associations.

3.1 Making object-oriented structures flexible

As pointed out above, the prerequisites for ontogenic adaptation are interchangeability of components and plasticity of structure. Traditional object-oriented systems do not fulfill both these requirements because 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 modeled in the design (e.g. in UML, roles are an annotation of an association), explicit representation of roles is lost in the implementation. Even though the specific interactions between objects are modeled using sequence and interaction diagrams, the *organization* expressed by those associations is not explicitly modeled in the design or apparent in the code. As well as being

implicit, the organizational structure can be difficult to change at design time and is largely frozen at compile-time.

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 of 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.

Design Patterns. While design patterns are usually viewed as repositories that facilitate design reuse, a number of them can be viewed as mechanisms for decoupling the structure of a program. The design patterns in [4] use configurations of objects that enhance the adaptability of a 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 behavior. The configuration of objects in the pattern gives a general solution to a problem in a way that promotes flexible design. As such, design pattern descriptions have an organizational aspect, although their scope is usually limited to fragments of design rather than being system-wide.

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.

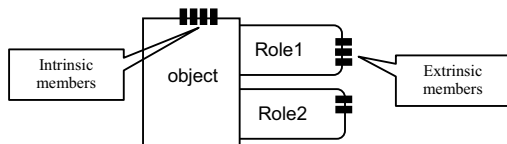


Figure 2 Object and role members

A number of authors propose that roles should be treated as first class modeling and programming entities [9,11,12,12]. Kristensen [11] 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 categorize 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 [8,11] encapsulate roles but allow 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 [12]. In this approach, all external behavior of the object should be encapsulated in roles. These roles are first-class entities with their own identity.

3.2. Organization of decoupled structures based on role associations

Organization in object-oriented systems can be viewed as a network of roles. Organizational descriptions need to express both a topology and a goal-oriented control regime for the whole system[22]. This role organization is a ‘global control-flow abstraction’[12]. Adaptable systems need to be able to change this topology and control regime. The Figure 3 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. Organizational descriptions are abstractions at the level of role associations.

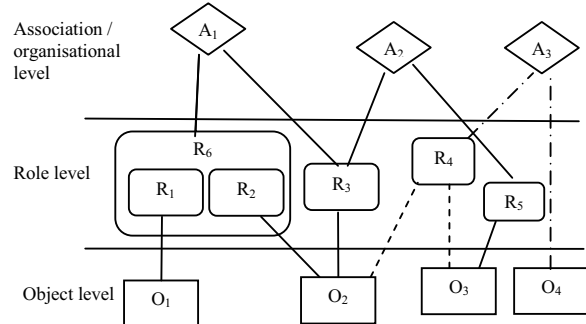


Figure 3 Associations in levels of abstraction in a role-based object-oriented system.

The diagram illustrates a number of possible types of association. Associations can be between single or aggregate roles (e.g. R₆). An object may adopt a number

of roles (e.g. O_2). If there is redundancy in the system, the same role might be performed by different objects (e.g. R_4 illustrated by the dotted line). Objects that have fixed behavior (such as resources) can be part of a role-object association (illustrated by the dash-dot line).

While design patterns and encapsulated roles may provide the prerequisite plasticity of structure, they do *not* provide global organizational abstractions. Some ‘system patterns’ [1], such as the compound Bureaucracy pattern [18] and SupplyChain patterns, do provide organizational abstractions. These can be considered proper, if primitive, organizational 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 organizational descriptions can be developed.

3.3. Separation of functional and management roles

In order to create organizational abstractions based on roles, we need to distinguish three types of role. These roles and their inter-relationship are shown in Figure 4 and explained below.

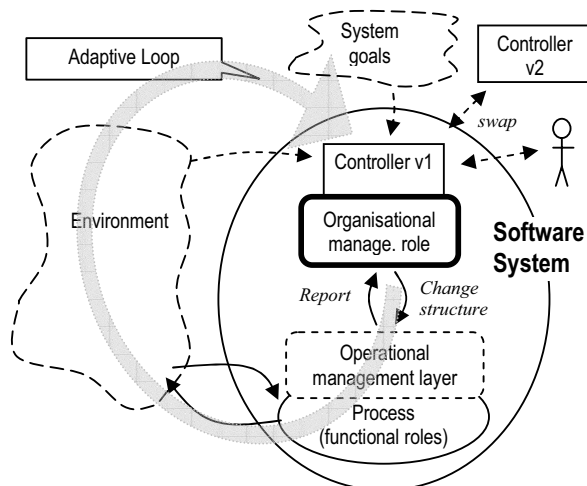


Figure 4 Overview: functional and management roles

- **Functional roles** 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** focus on regulating the relationships between functional roles. They define contracts between functional roles based on a conceptual separation of management control from process. The network of operational management roles represents the conduit for global control flow through the system (as

opposed to data flow). Operational management roles have no direct connection with the environment.

- **Organizational management roles** maintain a reflective representation of the system’s organization and mechanisms for restructuring the organization by creating/destroying role-object bindings and dynamic role-role associations. The controllers (objects/agents/humans) that play organizational management roles are responsible for the restructuring of 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*.

3.4 Functional roles

Classes in object-oriented software are commonly modeled on the basis of problem-domain responsibilities. As discussed in section 3.1, it is these extrinsic responsibilities are captured in roles. *Functional* roles express the processes of the system (including input and output to the environment), that achieve the system’s functional requirements. From our traditional object-oriented example in Figure 1, the Assembler subclass of the Employee class would become a functional role which has the responsibility of taking Thingy and Doover objects and making them into Widgets.

3.5. Operational-management roles

Few object-oriented systems 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 organizational framework. In our methodology we do this through a network of operational-management roles.

Operational management is the conduit for control flow through an existing organizational structure. In particular an operational manager would:

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

An example of an arrangement of operational management roles is the Bureaucracy pattern [18] illustrated in Figure 5 below. The Bureaucracy pattern is a compound pattern (or “system pattern” [1]) formed from the Mediator, Chain of Responsibility, Composite and Observer design patterns [4]. As well as their functional roles, objects can take on management roles within the organizational hierarchy of the system. The management-

type roles within the bureaucratic hierarchy include Clerk, Supervisor (in [18] called a “Manager”), Subordinate and Director. Every object within the hierarchy plays the role of a Clerk, plus either a Supervisor or Subordinate role 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 the Clerk and Director roles.

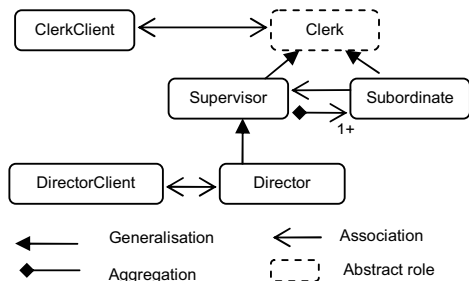


Figure 5 Bureaucracy Pattern - example organizational pattern based on operational management roles.

The management type roles listed above (Director, Supervisor, Subordinate etc.) are only one example of a group of roles that can form organizational 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 [9]. At courser levels of granularity, architectural styles such as master-slave, client-server and peer-peer can be viewed as expressing organizational relationships.

The separation of operational-management roles from functional roles gives us a way to describe the organizational structure of the system and the control regime of that structure. A definition of an abstract organization based on the bureaucracy pattern is shown in Figure 6 below.

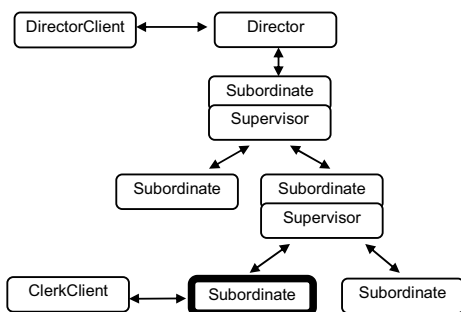


Figure 6 Abstract organization structure of operational-management roles based on the Bureaucracy pattern

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 organizational 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 organizational chart. A number of categories of topologies have been suggested in [22] for agent organizations. These could also be applied to object-oriented organizations:

- Collection of peers – small organizations 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

3.6. Operational-management contracts

Figure 6 is an abstract organization 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 organization was instantiated as a retail business, the object that plays the highlighted Subordinate role (in Figure 6) might also play the functional role of SalesPerson. Figure 10 in the next section gives an example of an instantiated organization.

Because every functional role would have a position in the organizational 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 organizational 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 [3] — 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 in Table 1 as follows:

Table 1 Example operational-management contract

Operational-management Contract		
Name	Supervisor-Subordinate	
Party A	Supervisor	
Party B	Subordinate	
A initiated	Invoke →	Accept
	Inform →	—
	Query →	Inform
	ResAlloc →	Accept
B initiated	Inform →	—
	Query →	Inform or Refuse
	ResReq →	ResAlloc or Refuse

When a functional role in an organizational structure is bound to an operational-management role using such a contract, 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 7 below.

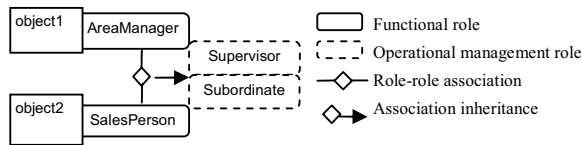


Figure 7 Association Inheritance

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 organization.

3.7. Organizational management roles

To be ontogenically adaptable, an organization needs mechanisms to effectively modify its organizational structure, either at design-time or run-time. We call this form of higher-level management *organizational management*, because what is being managed is the structure of the organization itself rather than the process. These roles maintain the organizational structure by creating and destroying object-role bindings and role-role association links. Organizational management ensures

system viability by both ensuring the internal organizational integrity, and monitoring the external system performance to ensure it is meeting its goals (NFRs). Organizational management responsibilities include:

- Allocation and de-allocation of functional roles to objects (instantiation of role through role-object binding).
- Aggregation / disaggregation of roles.
- Creating operational-management contracts and binding of functional roles to these contracts.
- Checking for organizational *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 external service discovery to fulfill roles in the organization.

Organizational management is separated into a role and a controller. The role maintains a representation of the organizational structure and mechanisms for manipulating the structure. The controller interacts with the interface provided by the role and the environment. As illustrated in Figure 4 above, the controller could be an object, agent, human operator or designer. It/he/she needs to have some understanding of viable organization (integrity rules and liveness/safety goals), and how to change the structure in order to achieve effective organization. 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 organization can be achieved through stages involving delayed binding under the control of a manager playing an organizational-manager role. These stages are:

1. Creating contracts between operational-management roles to form a network. This abstract organizational structure represents the topology of the organization 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 organizational structure.
3. The binding of those roles to objects. This creates on instantiation of the organizational structure.

4. An example

Returning to the Widget Factory example in the introduction, we will now show how a more adaptive design could be developed. To overcome the shortcomings of traditional object-oriented design, a

compositional role-object approach would decouple the class model in Figure 1. The classes ThingyMaker, DooverMaker, and so on, could be modeled as roles attached to an object of class Employee rather than as a specialization of Employee (through inheritance). Supervisor and Subordinate are *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 roles 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.

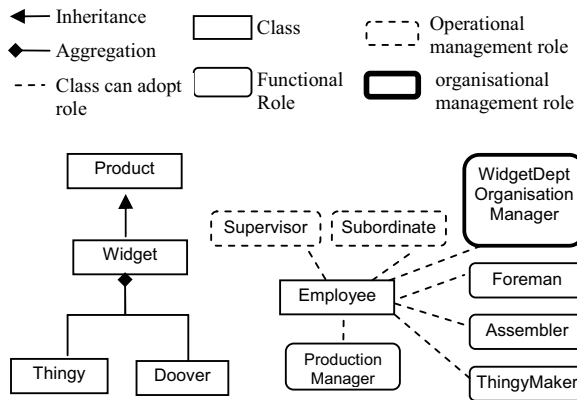


Figure 8 Partially Decoupled Class and Role Model

The topology of an organizational network based on Figure 8 is illustrated in Figure 9. This organizational structure is a cross-cutting aspect to the class structure.

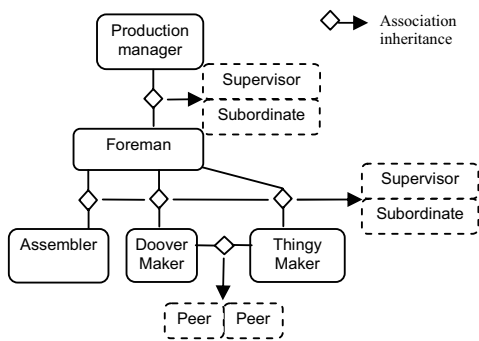


Figure 9 Abstract organizational structure

Note that the functional role associations inherit restrictions on interaction from the operational-management role associations. In Figure 9, the organizational structure is domain-specific but still abstract because, as yet, there are not object-role bindings.

By binding objects to the roles in the abstract organization we obtain a substantiated organizational structure. This is illustrated in Figure 10 below.

Employee *e4* performs two roles (DooverMaker and ThingyMaker) under the supervision of *e2* playing the role of Foreman. The Supervisor-Subordinate associations form a dynamically configurable network that enables the flow of control through the program. Under normal operating conditions, goal-oriented control flows through the organizational supervisor-control hierarchy.

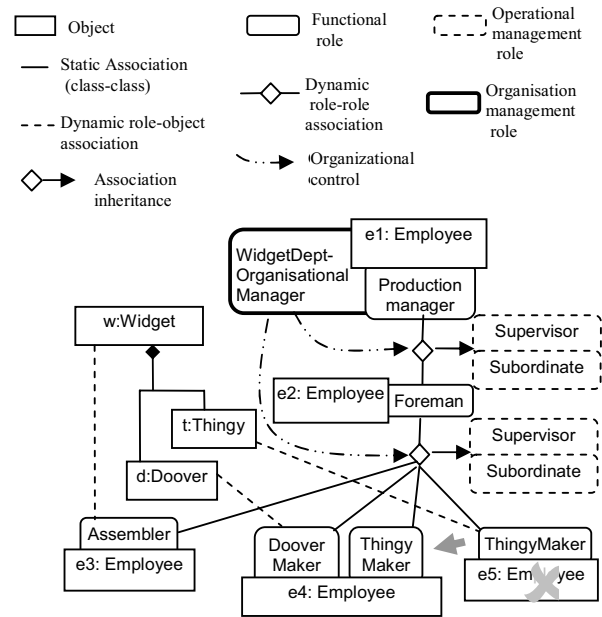


Figure 10 Instantiated organization

When the system needs to change its organization 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 organizational-management role which monitors the viability and performance of the organization in the adaptive loop. For example, suppose the computational node on which *e5* (in the figure above) runs becomes unavailable due to network failure. Employee *e1*, who plays an organizational-manager role, might transfer the role of assembler from *e5* to *e4*. Similarly, if a goal (non-functional requirement) of speed of production changes causing a bottle-neck because of the limited capacity of the *e3* object to carry out the Assembler role, additional Assembler positions could be added to the organizational structure.

Organizational management can be either centralized or distributed. If it is distributed to organizational managers, then their authority to alter the structure of the organization would have a limited scope. The role of an organization manager WidgetDeptOrganizationalManager

has been delegated to the object *e1* 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 organizational relations (object-role bindings and dynamic associations) in the diagram, *not* including its own role binding (*e1* cannot make itself a ProductionManager). For simplicity, the control links between the WidgetDept-OrganizationalManager and each of the object-role bindings have been omitted from the diagram. In the larger organization, distributing the roles of organization management across the system creates a degree of self-organization in the sub-systems or components.

5. Related work

Our methodology extends work on role and associative modeling in [2,8,9,11,12]. While these role-oriented approaches decouple the class structure, they do not define an organizational level of abstraction. Our method abstracts an organizational level through the separation functional and management roles.

Structural decoupling is also addressed in design patterns [2,4,18], although these patterns only describe design fragments. Another, more comprehensive, approach to decoupling is the Demeter method [13,14]. 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 organizational in that it has a method-focus rather than modeling software as a network of roles.

Organizational level abstractions are partially addressed in work on software architecture [1] and frameworks [19,21]. According to Garlan [5], 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 should be *abstract* expressions of the management responsibilities and the run-time control of components [1]. However, as with the limited role-oriented approaches mentioned above, there is no basis for encapsulating the management roles that represent a control regime. Architecture is reduced to a static representation of components and connectors which is not adaptable.

A number of possible mechanisms have been suggested[11] for adding roles to objects including *aspects*, *mixins* and the *decorator* pattern [8]. Kendall [9] 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. As no distinction is made between functional and management

roles in previous work, a differentiated approach to implementation has not been considered.

Work on roles has also been undertaken in multi-agent software engineering [7,16,22]. In particular, [22] extends the concept of a role model to an organizational 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 [21]. Such systems are designed to maintain system viability during anticipated environmental perturbation, but they cannot be considered adaptive. Recent work on intelligent control [6] adds an adaptive loop on top of the operational control loop. This is similar to our concept of organizational-management roles that control the structure of the organization 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 organizational 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 organizational structure that represents the topology of the organization 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 organizational structure. An instantiated organizational 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 *organizational-management* roles. These roles control and maintain the organization by creating and destroying object-role bindings and dynamic role-role associations.

Before the efficacy of this approach can be tested through case studies, further work needs to be done on appropriate implementation mechanisms for encapsulating functional, operational and organizational 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 as in [8], then organizational 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 be suitably implemented as aspects, as the contracts they define are relatively stable. The permissions defined in those contracts crosscut the functional roles at various join points (e.g. method invocation). Finally, the domain of organizational-management roles is the organization itself rather than the problem domain. Some explicit representation of roles and their binding to objects is necessary. Although organizational roles are not in themselves a cross-cutting concern, the monitoring of the organization 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 [14] and Service Model from Gaia [22] may provide general solutions to compatibility between such objects and roles.

The nature of operational management needs to be further developed. Associations other than supervisor-subordinate need to be examined and contract protocols developed for all such associations. It may be possible to develop a pattern language of such operational management associations. Kristensen [10] 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 organizational structure which will allow the maintenance of organizational stability. Control-theoretic approaches [6,20] to building organizational controllers may suggest useful approaches for adding the necessary dynamic environmental coupling.

7. References

- [1] Bass, L., Clements, P., and Kazman, R. *Software architecture in practice*, Reading, Mass: Addison-Wesley, 1998.
- [2] Bäumer, D., Riehle, D., Siberski, W., and Wulf, M. "Role Object" in *Pattern languages of program design 4*, eds. Harrison, N., Foote, B., and Rohnert, H. Addison-Wesley, 2000, pp. 15-32.
- [3] The Foundation for Physical Intelligent Agents, *FIPA Communicative Act Library Specification* <http://www.fipa.org/specs/fipa00037/>, 2002, last accessed 27 Aug 2004
- [4] Gamma, E., Vlissides, J., Johnson, R., and Helm, R. *Design patterns elements of reusable object-oriented software*, Reading, Mass: Addison-Wesley, 1995.
- [5] Garlan D., "Software architecture: A roadmap." *Proceedings of the International Conference on Software Engineering: On the Future of Software Engineering, Limerick, Ireland, 2000*; 2000, pp. 91-101.
- [6] Herring, S. and Kaplan, C., "Viable Systems: The Control Paradigm for Software Architecture Revisited" *Australian Software Engineering Conference. 2000*, 2000, pp. 97-105.
- [7] Juan, T., Pearce, A., and Sterling, L., "ROADMAP: extending the Gaia methodology for complex open systems" *Proceedings of the first international joint conference on Autonomous agents and multiagent systems, Bologna, Italy, ACM, 2002*, pp. 3-10.
- [8] Kendall, E. A., "Role model designs and implementations with aspect-oriented programming." *Proceedings of the ACM Conference on Object-Oriented Systems, Languages, and Applications, Denver, CO, 1999*, pp. 353-369.
- [9] Kendall, E. A., "Role Modelling for Agents System Analysis, Design and Implementation" *First International Symposium on Agent Systems and Applications IEEE CS Press, 1999*
- [10] Kristensen, B. B., "Associative Modeling and Programming." *Proceedings of the 8th International Conference on Object-Oriented Information Systems (OOIS'2002), Montpellier, France, 2002*
- [11] Kristensen, B. B. and Osterbye, K., "Roles: Conceptual Abstraction Theory & Practical Language Issues" *Special Issue of Theory and Practice of Object Systems (TAPOS) on Subjectivity in Object-Oriented Systems, 1996*
- [12] Lee, J.-S. and Bae, D.-H., "An enhanced role model for alleviating the role-binding anomaly" *Software: practice and experience*, vol.32, 2002, pp. 1317-1344.
- [13] Lieberherr, K., Orleans, D., and Ovlinger, J., "Aspect-Oriented Programming with Adaptive Methods" *Communications of the ACM*, vol.44(10), 2001, pp. 39-41.
- [14] Lieberherr, K. J. *Adaptive object-oriented software the Demeter Method with propagation patterns*, Boston: PWS Pub. Co, 1996.
- [15] Maturana, H. R. and Varela, F. J. *Autopoiesis and cognition the realization of the living*, Dordrecht, Holland, Boston: D. Reidel Pub. Co, 1980.
- [16] Odell, J., Parunak, H. V. D., Brueckner, S., and Sauter, J., "Changing Roles: Dynamic Role Assignment" *Journal of Object Technology, ETH Zurich*, vol.2(5), 2003, pp. 77-86.
- [17] Parunak, V. and Brueckner, S., "Engineering Self-Organising Applications (Tutorial 5)" *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent System (AAMAS'03)*, 2003, pp. 1-54.
- [18] Riehle, D. "Bureaucracy" in *Pattern Languages of Program Design 3*, eds. Martin, R., Riehle, D., and Buschmann, F. Reading, Massachusetts: Addison-Wesley, 1998, pp. 163-186.
- [19] Riehle, D., *Framework Design - A Role Modeling Approach*. Dissertation, Swiss Federal Institute of Technology, Zurich. <http://www.riehle.org/computer-science/research/dissertation/>, 2000.
- [20] Shaw, M., "Beyond objects: A software design paradigm based on process control." *ACM Software Engineering Notes*, vol.20(1), 1995, pp. 27-39.
- [21] Shaw, M. and Garlan, D. *Software architecture perspectives on an emerging discipline*, Upper Saddle River, N.J: Prentice Hall, 1996.
- [22] Zambonelli, F., Jennings, N. R., and Wooldridge, M., "Developing multiagent systems: The Gaia methodology" *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol.12(3), 2003, pp. 317-370.