

# Roles, players and adaptable organizations

Alan Colman<sup>a,\*</sup> and Jun Han<sup>b</sup>

<sup>a</sup> Faculty of Information and Communication Technologies, Swinburne University of Technology,  
PO Box 218, Hawthorn, 3122, Melbourne, Victoria, Australia

Tel.: +61 3 9214 8771; Fax: +61 3 9819 0823; E-mail: acolman@it.swin.edu.au

<sup>b</sup> Faculty of Information and Communication Technologies, Swinburne University of Technology,  
Melbourne, Victoria, Australia

E-mail: jhan@it.swin.edu.au

**Abstract.** Role is a commonly used concept in software development, but a concept with divergent definitions. This paper discusses the characteristics of roles in software organizations, and contrasts such organization roles with other player-centric conceptions. Roles in organizations have their own identity and do not depend on role players for their existence. In software terms, such roles are first-class runtime entities rather than just design concepts. We define characteristic properties of both roles and players in organizational contexts, and show how the boundary between a role and its player varies depending on the level of autonomy the player is allowed. We show how roles can facilitate the separation of structure from process facilitating greater adaptivity in software. The problem of preservation of state in role-based organizations is also discussed. Possible implementation strategies for both roles and players are discussed and illustrated with various role-oriented approaches to building software organizations.

## 1. Introduction

In its general usage, the notion of role has been introduced to account for the relationships of an individual within a particular social context (Steimann, 2000). In this paper, we are concerned with software systems in which social contexts are intentionally designed and structured. We call such contexts *organizations*, where an organization refers to both the role relationships in the system, and the processes that maintain the viability of these relationships in response to changing goals and changing environments. Roles are the nodes of designed organizational structures. Software systems built on such structures can be both *goal-directed* and *adaptive* (Colman & Han, 2005a).

If we are to build such role-based software structures, we need to define the essential properties of roles in software organizations, and how these properties are distinct from the properties of the software entities that play these roles. In this paper our focus is on the binding between *organizational roles* and the players that play them, and on the shifting boundary between role and role-player. Players in different roles within an organization may require different degrees of autonomy and capability. They may be objects, agents or humans. Indeed, we have argued elsewhere (Colman & Han, 2005b) that differentiated autonomy and capability is an essential characteristic of complex role-based systems. As we will show later in the paper, such capabilities do not necessarily, for example, include the reasoning capability associated with proactive agents. While we do not elaborate the concept of (software) organization in

---

\*Corresponding author.

this paper, we need to have a clear conception of the characteristics of the roles and players if we are to begin to apply organizational abstractions to the domain of software.

The paper is structured as follows. Section 2 is a brief discussion of the concept of roles in software development methodologies. In particular, we examine whether methodologies use roles merely as an analysis/design concept, or whether the role is reified as an implementation entity. Section 3 addresses the following questions. If roles are implementation entities, to what degree are roles and their players separate? Do roles have an independent identity? Can a role exist independently from the role player? We will briefly examine the various methodological responses to these questions, and point to the need to radically separate roles from players in organizational structures. We propose that role identity should be organization-centric, rather than player-centric. Section 4 examines the essential properties of both roles and players within an adaptive organization. If we want to design and implement an explicit organizational structure, what properties will be needed in the roles that make up that structure? Consequently, what general properties will be needed by the players who play such organizational roles? The subsequent sections of the paper discuss the issues that arise from the radical separation of roles and players, and discuss how various approaches to implementing software organizations have addressed, or might address, these issues. Section 5 proposes a conceptual framework based on the autonomy permitted by the role and the capability of the player. Players in different roles within a software organization may be very heterogeneous (objects, components, services, agents or humans), and have very different degrees of autonomy and capability. We then examine various implementation strategies for defining role-players with various levels of autonomy. How do we ensure that the player is capable of playing the role assigned to it? Possible solutions to this problem are discussed and related to object-oriented and agent-oriented approaches. Section 6 proposes that players should use “blind communication” in order to make organizations more adaptable through maintaining the separation of organizational structure and process. Section 7 discusses the problem of the preservation of state in role-based organizations. We conclude with Section 8.

## **2. Roles as design and implementation entities**

Roles are a recurring concept in both object-oriented and agent-oriented methodologies, not to mention data-modeling (Steimann, 2005). In conventional object-oriented methods, roles have figured as a classifier of the relationship between objects. In UML, roles are a descriptor for the ends of an association between classes (the concept of role has been subsumed by the concept of ConnectorEnd in UML 2.0 (Object Management Group, 2004)). In some methods, such as OOram (Reenskaug, 1996), roles are central concepts to the analysis and design. In OOram roles are nodes in an interaction structure (role-model). These role-models can be based on any suitable separation of concerns. Responsibility-driven design (RDD) also focuses on collaborations between roles, but such contracts between roles are seen as “really meaningful only in the programmer’s mind” (Wirfs-Brock & McKean, 2002). In such approaches roles are used in the modeling and to inform the design, but disappear as entities during implementation.

Other approaches based on role and associative modeling define roles as first-class design and implementation entities (Kendall, 1999a; Kristensen & Osterbye, 1996; Lee & Bae, 2002; Fowler, 1997; Bäumer et al., 2000). Fowler (1997) discusses the implementation of roles in object-oriented design using a variety of object-oriented patterns. Kendall (1999b) has shown how aspect-oriented approaches can be used to introduce role-behavior to objects. In Kendall’s approach, roles are encapsulated in aspects

that are woven into the class structure. Such approaches see roles as encapsulated implementation entities, but they vary as to whether roles can exist independently from the objects that play them. A number of object-oriented frameworks and languages that treat roles as first class entities have been developed (Baldoni, Boella & van der Torre, 2006b; Colman & Han, 2005a; Herrmann, 2002). These are discussed in more detail below.

Roles also figure in a number of agent-oriented approaches (Dastani, Dignum & Dignum, 2003; Juan, Pearce & Sterling, 2002; Ferber & Gutknecht, 1998; Odell et al., 2003; Zambonelli, Jennings & Wooldridge, 2000). Gaia in particular (Zambonelli, Jennings & Wooldridge, 2003) extends the concept of a role model to an organizational model. In these approaches, like some object-oriented approaches, roles are not an implementation entity. For example, in Gaia role models are developed at the analysis and architectural design stages, but roles are mapped to agents (not necessarily on a one-to-one basis) during the detailed design stage. Other agent-based models (Odell et al., 2004) see roles as a key modeling concept but, being implementation-independent, these models give no indication of how these roles are to be realized. For Dastani et al. (2003), roles are formal specifications that can be checked at runtime to ensure that agent specifications are compatible with those roles. In general, if an agent-oriented methodology only has agents available as implementation entities, then it will lack the expressiveness to explicitly represent roles in an organizational structure.

As our aim is to create explicit organizational structures at the implementation/code level, we require roles that can be created and manipulated as implementation entities.

### 3. Two perspectives on roles – player and organization

Kristensen (1996) defines the characteristics of roles in object-oriented modeling. These include:

- *Dependency*: A role cannot exist without an object;
- *Identity*: An object and its role have the same identity;
- *Visibility*: The visibility and accessibility of the object (a.k.a. player) is restricted by the role;
- *Dynamicity*: A role may be added or removed during the lifetime of an object;
- *Multiplicity*: Several instances of a role may exist for an object at the same time;
- *Abstractivity*: Roles can be classified and organized into generalization and aggregation hierarchies.

The above characterization of a role has been widely adopted in the object-oriented literature, if not in the object-oriented practice. Roles can be implemented as runtime entities and yet have no independent identity or separate existence from their players. For example, Kendall (1999a) and Kristensen (1996) encapsulate roles as implementation entities but allow them no existence separate from the objects to which they are bound. While roles exist as a class, they can only be instantiated when bound to an object. Steimann (2000) provides a useful overview of approaches where roles are seen as adjuncts to object instances. Roles are seen as clusters of extrinsic members of an object. Such roles are carriers of role-specific state and behavior but do not have an identity.

All the above approaches are object-centric or *player-centric*. The object is seen as the stable entity to which transient roles are attached. The identity of the role is an adjunct to the identity of the object. The role of an object is not a different entity, but merely its appearance in a given context (Steimann, 2000).

An alternative perspective, which is adopted in this paper as well as in Baldoni, Boella and van der Torre (2005), Baldoni, Boella & van der Torre (2006b), Herrmann (2002) and Herrmann (2005), is to look at roles from an organization-centric viewpoint. From this perspective, a role's identity and

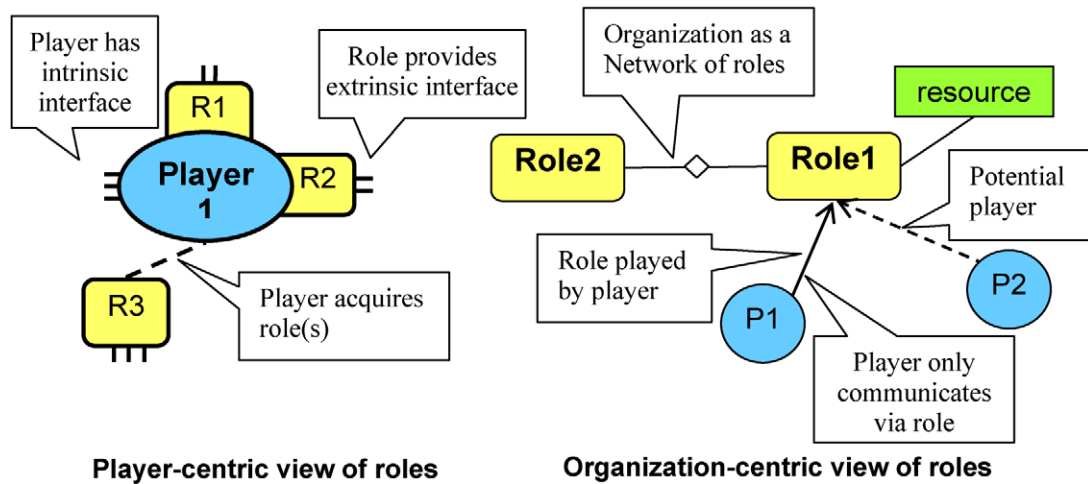


Fig. 1. Player-centric and organization-centric perspectives on roles.

existence derives from the organization that defines the roles and associations – not from the player itself. This dependency of a role on a group is also apparent in some agent-oriented approaches (Odell, Nodine & Levy, 2005). Roles are the more stable entity within the organizational structure and transient players are attached to these roles. A role instance<sup>1</sup> may be played by different players at different times (although not simultaneously). In an organization, there is generally no restriction on a single player playing multiple roles. These two perspectives are illustrated in Fig. 1.

The organization-centric view of roles accords more with the characteristics of roles in human organizations, such as a bureaucracy or a business. Where such organizational roles are part of the organization's process, we call them *functional* roles. When the organizational role's purpose is to reconfigure or regulate (manage) the organization itself, we call them *organizer* roles. In this organizational view of roles, the role *defines* a process (at some level of abstraction), and the player is responsible for *executing* the process. This view sees roles and players on different ontological levels. The *player-as-executor* in this conception of organizational roles, is fundamentally different from player-centric conceptions that view of the relationship between a role and a player as an adjunct instance (the execution of the process is shared between role and player), or as an inheritance relationship (a role is a specialization of the player) as found in some role modeling approaches (e.g., Paesschen, Meuter & D'Hondt, 2005).

As relationships are defined on roles rather than players (or entity types (Steimann, 2005)), the network of these roles is the basis of the organizational structure. If functional roles are nodes in an organizational structure, then a role may have associations with more than one other role. These associated roles may also be of various role-types. A functional role may therefore have a number of interfaces – one for each of its association types. This is a different view from a conventional object-oriented view of a role: as a descriptor for one end of a single association.

<sup>1</sup>We use the expression "role instance" to denote a specific placeholder within an organization, as distinct from the role-player that executes the role. Organizational roles are defined by their relationships (obligations and powers) with other roles within the organization, rather than being defined by any intrinsic property of the role itself. It follows that roles can only be generalized as 'role types' to the extent that roles of the same 'type' share the same types of relationship with other role types. In our view, role instances are therefore not strongly typed, in the way that, in an object-oriented system, objects are instances of a class type with intrinsic attributes. For a more detailed discussion of the various conceptions of role and class (natural type) hierarchies see Steimann (2000).

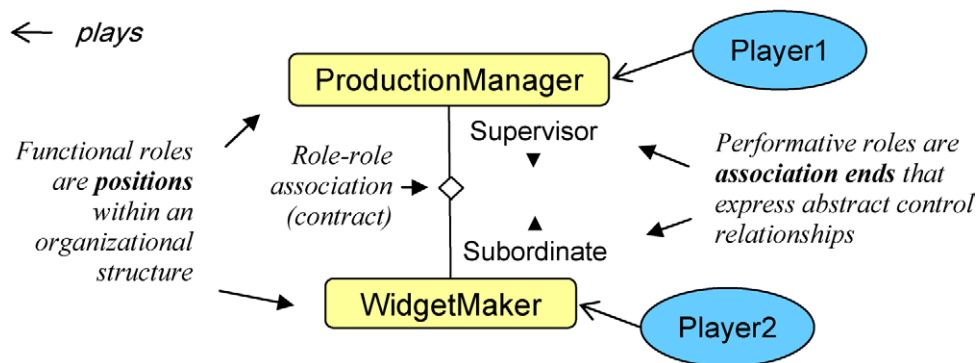


Fig. 2. Functional and management roles.

Loebe (2005) in his top-level ontological analysis of the role concept defines three types of role: *relational*, *processual* and *social*. From the above discussion we can see that functional roles in an organization are *processual* (in that they define the process at some level of abstraction) and *social* (in that they are their identity depends on the organization). Functional roles in an organization participate in a process and they belong to an instance of a social context (the organization).

In an organization, the *associations* between functional roles can also be characterized. Elsewhere, we (Colman, 2006) have characterized the control aspects of these role associations, the ends of which we term *performative roles*. Such roles specify the *type* of message one role-player can send to another role within the same organization (for example, an agent may ask another, via its role, to *do* something; or provide information; etc.). Performative roles are classifiers for the ends of an association, and always come in complementary pairs as shown in Fig. 2. As such, they are instances of Loebe's (2005) other top-level classification: *relational* roles. Examples of performative role-pairs include supervisor–subordinate, auditor–auditee, predecessor–successor, buyer–seller, and so on. Such performative relationships define patterns of interaction between roles at an abstract level but, unlike functional roles, do not serve as a position in an organizational structure. We have shown elsewhere (Colman & Han, 2005c) how contracts that bind functional roles can inherit such patterns of interaction from abstract performative contracts. In this paper, we restrict our discussion to functional roles and take the word “role”, unless otherwise stated, to refer to functional roles within an organization. The relationship between functional and performative roles is illustrated in Fig. 2.

A functional role is a “position” to be filled by a player (Odell, Nodine & Levy, 2005). There may be multiple roles of the same type within an organization. Some of these roles may be temporarily unassigned to players. For example, if an employee (player) resigns from her role as Production Manager within a manufacturing business, the role does not cease to exist. That position (role) within the company structure may be temporarily unassigned, but the company may continue to function viably in the short term. Orders can still be taken, current work orders still be manufactured, finished goods can still be shipped, accounts can still be processed and so on. Nor does the identity of the role depend on the identity of the player. From the organization's point of view it does not matter whether employee John Doe or Jane Smith performs the role of Production Manager as long as they both have sufficient capability. In a viable organization the role model (organizational structure) is not just a design concept that helps to structure the relationships between employees (players). It is also a set of relationships between roles that is maintained and manipulated (to some degree) independently from the players that are assigned to

those roles. The ability to dynamically bind the different players to a role gives the organization a degree of adaptability in meeting changing goals and environments.

Organizational roles therefore can be thought of as having a number of states in relation to the binding with a player. A specific role can either be assigned to a player or left unassigned. As Odell et al. (2003) point out, the relationship between a role and an assigned agent may also be in an active or suspended state (e.g. our Production Manager has gone to lunch and although not active in her role still occupies that position). Likewise, Dastani et al. (2004) propose a similar set of concepts to define the status of a relationship between a role and an agent-player, and then formally define a set of operations to change this status.

To summarize the characteristics of functional roles within an organization, Kristensen's characteristics of *Dependency* and *Identity* do not hold. For organizational roles we can modify these characteristics of roles as follows:

- *Existence independent of player*: Roles in an organization do not depend on players for their existence. They depend on the organization for their existence.
- *Independent identity*: Roles have an organizational identity that is independent from their players even though the role and player constitute a unity within the organization.

The other characteristics are still applicable. The characteristic of *visibility* holds for roles because, in an organizational context, players always interact with each other via their roles. However, a further question arises as to whether or not *players* bound to an organization are *invisible* to each other. In player-centric approaches to organization, players can each see other because role and player have the same runtime identity. For example, in agent-oriented approaches, where agent-players internalize their role specifications (e.g., Dastani, Dignum & Dignum, 2003), the players are still visible to each other, albeit through the filter of a role. In organization-centric approaches, where roles are reified as runtime entities and all interaction occurs through those role entities, it may be that players are unaware of each others identity. This is the approach taken in Colman (2006) where roles are the nodes of an organizational structure over which all interaction within the organization occurs. In this case, the player is aware of its own role, but cannot see other functional roles (or their players) in the organization. As discussed in Section 6, it is the visibility of one functional role to another that defines the organizational structure.

The separation of organizational roles from the entities that play them allows the definition of abstract organizational structures that are independent from particular players. Such a structure in a human organization would be described, for example, in a company's organizational chart where the nodes are the roles in the company and the arcs are the authority relationships. However, the radical separation of roles from role-players introduces the problems of how to define the dividing line between extrinsic (role) and intrinsic (player) properties in the combined role playing entity; how to ensure role-player compatibility; and how to preserve the integrity of the organization's processes when players are swapped. We address these issues in the following sections.

#### 4. The properties of roles and players in adaptable organizations

We conceive of the role as expressing a function that serves some purpose in the organization. It defines what the role-player needs to do at some level of abstraction, and defines the provided and required relationships with other roles. It may also define a relationship to tools and resources. The player, on the other hand, initiates actions in line with its capability to perform the defined role. The player has

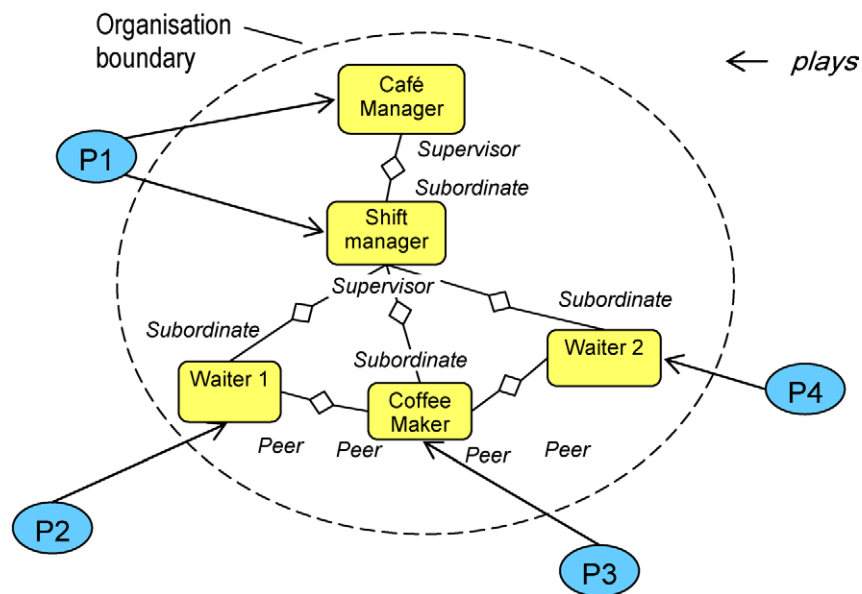


Fig. 3. Organizational chart with players.

intrinsic properties that give it this capability. This concept of the intrinsic nature of player capability is common to both object-oriented (“core object” (Kristensen & Osterbye, 1996)), and agent-oriented approaches (“agent physical classifier” (Odell, Nodine & Levy, 2005)).

Let us illustrate the separation of properties between a role and a role player with an example from an organization made up of people – a coffee shop business that has a role of coffee-maker. The organizational context (the coffee-shop business) of this role is illustrated in Fig. 3.

The coffee-maker role might be defined as follows. The goal associated with the coffee-maker role is to make quality coffee to a certain standard within certain time constraints, in response to requests from waiting staff. The role defines work instructions for preparing the coffee to the business’ standard. The role also gives access to resources such as the espresso machine and ingredients. The role defines functional relationships with other roles in terms of what is provided and required. It also defines authority relationships with other roles in the business. For example, the coffee-maker is subordinate to the shift manager, peer to the waiting staff and so on. These authority (performative) relationships define the valid types of control communication that can pass between actors playing the respective roles. A role has (explicitly or implicitly) a “position description” that describes the capabilities needed of a player assigned to play that role. In order to effectively play the role of coffee-maker, an employee (or rather a person playing the role of employee which is itself a generalization of the coffee-maker role) needs to be able to follow work instructions, use the tools required for the role to transform the ingredients as required, and communicate with other role players following the conventions imposed by the authority relationships between the role types.

Note that although roles can be duplicated (e.g. Waiters) each role instance has at most one player assigned to it at any one time. However, there may be no impediment to a role player performing multiple roles in an organization (e.g. employee P1 in Fig. 3).

#### 4.1. Properties of organizational roles

An organizational role has the following properties.

- The *function* of the role expressed in terms of purpose, system-state, or process descriptions (depending on how detailed the level of prescription in the role and how much autonomy the player is able to exercise).
- *Performance requirements* for executing its functions are a property of the relationship of the role with its enclosing organization, rather than an essential property of the role itself. The role's performance requirements are therefore an aggregation of the performance requirements of all its relationships. Such performance criteria are set by the organization. Actual performance of a role is always an externally measurable property of an assigned role (player-role pair) because different players may have varying capability in performing the role. More correctly, performance of a role-player pair is always *in relation* to its organization, perhaps as represented by other roles in that organization. It is therefore appropriate to represent performance as a property of a relationship rather than an entity. For example, in Colman (2006), actual performance (non-functional requirements) is recorded within contracts that associate roles.
- *Interaction protocols* and *authority relationships* (power, expectations and obligations) with respect to other roles within the organization and with external entities with which the organization has associations.
- Access to, and restrictions on, *resources* controlled or owned by the organization.

The above properties are fundamentally properties of the role's relationship with its organization, i.e. other roles and resources in the organization. The role description is an aggregation of the properties derived from its relationships. From the perspective of a role player, this aggregate description is a definition of the knowledge and skills required in a player to enable performance of the role function. This is an interface definition with both functional and non-functional requirements. A role player needs to be able to execute the function defined in this interface at the specified level of performance and while meeting any other non-functional requirements it defines. We discuss the nature of the role-player interface in the next sections.

#### 4.2. Properties of organizational role-players

A role player who is bound to a role in an organizational structure needs an ability to perform the assigned role. Depending on the level required this capability might include:

- Execution of function defined by the role or imposed by role relationships, including the ability to use tools and resources provided by the role.
- Ability to communicate, *via its role*, with other roles within the organization or with external entities (if so required by the role).
- If required by the role, the player may need to have reasoning ability concerning what processes can achieve given system-states, or reasoning ability about which system-states to best achieve the role's purpose. Consequently, an ability to perceive external states to assist in the reasoning process may be required.

A role and its player act as a unity within the organization, even though roles within an organization have an existence and an identity that is independent from their players. If we are to implement roles as

first-class entities in a runtime organization, a number of issues arise from the radical separation of roles and players. These issues relate to the division of responsibility between roles and players, namely:

- What level of autonomy do players have in fulfilling their role? Do roles “do” anything?
- Does knowledge of the organizational structure reside in the role, player or both?
- Are the roles or are the players responsible for maintaining their state within the organization?

Sections 5–7 discuss these issues and various strategies for the implementation of roles. The purpose of this discussion is to point to the range of ways organizational roles might be implemented, rather than set out a definitive prescription of how roles should be implemented. In particular, we will (where appropriate) illustrate these issues by comparing and contrasting how they have been implemented in three different approaches to creating role-oriented software organizations. These are powerJava (Baldoni, Boella & van der Torre, 2005; Baldoni, Boella & van der Torre, 2006b), ObjectTeams (Herrmann, 2002; Herrmann, 2005), and our Role-Oriented Adaptive Design (ROAD) framework (Colman & Han, 2005a; Colman & Han, 2006).

#### 4.3. Role-based organizations

Before discussing the issues associated with the separation of organization-centric roles and players, we need to outline the relationships of these entities to the organization itself. In addition to defining and creating role entities, an organization (or rather a manager within the organization) is responsible for creating the relationships between its roles, for checking the compatibility between roles and players, and creating/destroying the binding between a role and its player. The binding of a player to an organization-centric role is more analogous to a person being employed by a business organization, rather than a social relationship between two free-agents. While agreement must initially be reached between the organization and the player, it is the organization that ultimately defines the binding.

An organization *is* a type of role-player. At runtime, an organization has its internal roles instantiated with players. This composite of connected roles and their players is encapsulated and executes a goal-oriented function. In this way, recursive compositions of role composites can be formed. Again, this is analogous to the way business organizations are structured, with business playing roles (e.g. supplier) with respect to other organizations. Internally, a business can be recursively decomposed into a set of inter-related roles, some of which can be played by role composites (say, a department within the business). Interactions between composites are always via well-defined interfaces that are defined in the roles they play. Adaptivity of the organization is improved because the player is always separable from its role, and can be replaced if a viable alternative exists. In our analogy for example, the functions of the department could be outsourced to an external organization. This ability to separate a composite-player from its role does *not* imply that a composite necessarily presents only one “whole-of-business” interface. A composite, like any player, can play a number of roles and can present interfaces appropriate for each of these roles, as shown in Fig. 4.

Figure 4 illustrates the various relationships between roles and composites, and how role-composites can play roles in other composites. For the purposes of illustration we will suppose the composites are encapsulated as Web services. Composite A has three internal roles, and the composite presents a Web service interface for each of these. One of these roles, A3, is currently played by a service Composite B. In this case, all of Composite A’s messages to Composite B pass through its role A3. Composite B, on the other hand, must include some form of message routing mechanism so that messages can be passed between the *external* role (A3) and the appropriate *internal* role (B1 | B2 | B3). In Composite B,

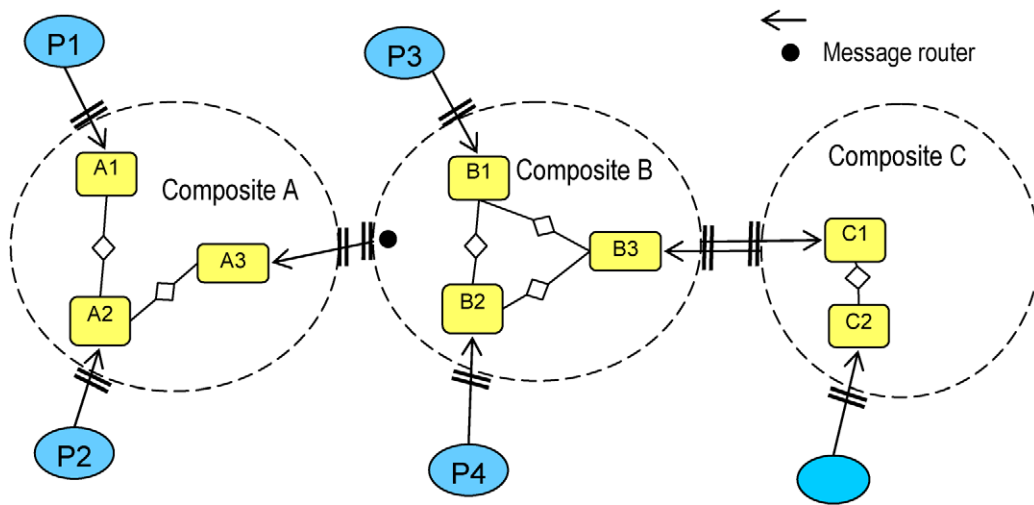


Fig. 4. Organizations as players that encapsulate role composites.

the relationships between B's interface to A3 and its internal roles is not explicitly modeled in the role structure, but is written into the message router.

An alternative approach is illustrated in the relationship between Composites B and C. In this case, each composite has a role that represents the other composite. Composite B plays the role C1 in C, and C plays the role B3 in B. Each composite can therefore explicitly model and easily control its own binding to the other. This arrangement would be beneficial in open systems where B and C may have different owners.

The organizational paradigm provides a way to think about software at a higher level of abstraction than the partial perspectives of structure or behavior. While *structures* are generally conceived of as static with fixed behavior, the concept of *organization* suggests the ability to restructure in order to meet changing goals and changing environmental conditions. In this sense, an organization is not only a collection of related roles but it also has adaptive behavior that maintains its viability in uncertain environments. If necessary an organization can restructure itself to improve its performance. This restructuring can involve the creation of new roles or the redefinition of existing roles. These new roles may perform the same function as roles that already exist, or perform different functions. Players can be assigned and separated from roles. Such organizations typically have layers of management to control its operations and restructure if necessary.

In the ROAD ontology, roles are composed into self-managed composites under the control of an organizer. These organizers are themselves a type of organizational role played by a 'manager' player. The difference between a *functional role* and an *organizer role* is the scope of control: a functional role controls a domain-process while the organizer role controls the organization of its own composite. Power to act within the composite is conferred by its organizer who creates contracts between the roles. When multiple composites are composed, their organizer roles form a management network *between* composites (analogous to a nervous system) that is separate from the network between functional roles *within* a composite (Colman, 2007). An in-depth discussion of organizations as managed role-based composites can be found in Colman (2006).

### 5. Levels of player autonomy

Complex systems can comprise heterogeneous players with varying capabilities. The degree of control that the role imposes on its player may vary depending on the amount of autonomy that the organization allows the player, and on the level of capability of the player. Returning to our example of a coffee-maker, the work instructions for making coffee may vary in terms of their level of detail. Inexperienced coffee-makers may require detailed instructions on how to make a cup of coffee, while an experienced and capable coffee-maker may not need to follow instructions defined by the role but may just be given a system-state (“strong cappuccino”). This experienced coffee-maker may alter the process depending on the inputs (“the coffee-maker perceives the beans to be a darker roast than usual”).

As can be seen from the above example, the granularity of the descriptions of the task associated to a role may vary depending on the capability of the player. Intentional action can be described at various levels of abstraction on a means-end (intentional) hierarchy as shown in Fig. 5. In an intentional hierarchy, goals are operationalized at successively lower levels of abstraction, and the purpose of a level of abstraction can be determined by referring to higher levels of abstraction. At its most abstract level, the role to be performed may be described by a *purpose* or *goal* in the environment external to the role or organization (“keep the customer’s happy by making good coffee”). At a more detailed level, the means to achieving the goal of the system might be described as a *state* of the system itself (“make a coffee to standard X”). At the next level of operationalization, the *process* for achieving that state would be described by the role (“follow the coffee-making work-instructions”). Such work instructions could then be described by the role at progressively more detailed granularity.

At some point the atomic goals/states/processes must be able to be interpreted and executed by the player. In this sense, the relationship between role and player is more like the relationship between program and abstract machine, than between two components at the same level of abstraction.

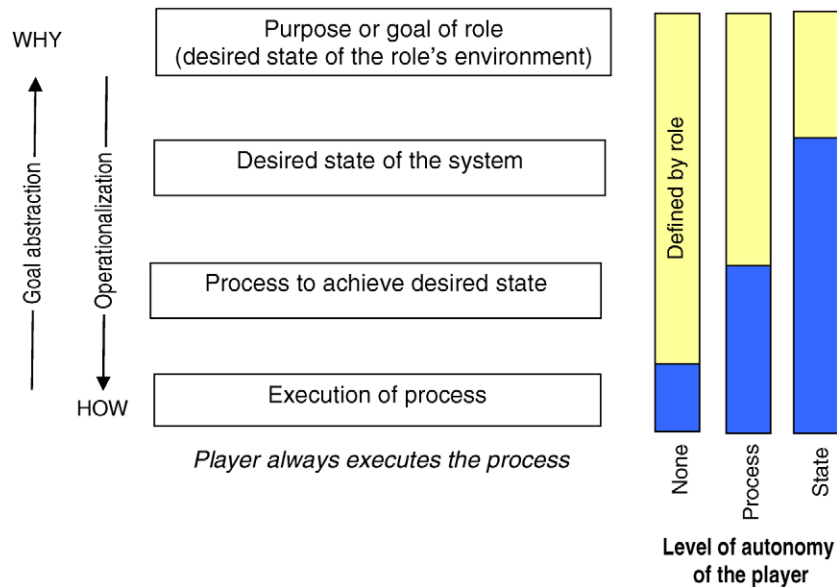


Fig. 5. Shifting boundary between roles and players on an intentional hierarchy.

In an organizational context, the amount of autonomy that a player can exercise in a role *is determined by the organization rather than the player*. Based on the above intentional hierarchy, we can identify five levels of player autonomy. Ordered in terms of increasing player autonomy, these are:

1. **No autonomy** – player executes the process defined in the role.
2. **Process autonomy** – player can choose process to meet system state defined in the role.
3. **System-state autonomy** – player can choose a system state and processes to fulfill a goal defined by the role.
4. **Intentional autonomy** – player can choose if it will fulfill a goal/state/process defined by the role. Such autonomy, to choose whether or not to adopt a goal, is orthogonal to degree of autonomy a player is given in operationalizing a goal (levels 1–3).

An additional level of autonomy can be identified, although it is not one defined in the role, that is:

5. **Autonomy from constraints** – player can violate constraints defined in the role.

As the role is organizationally defined, the intention or purpose of the role is always defined external to the player (implicit in the role itself). On the other hand, the process is always executed by the player. In this conception, unlike the player-centric view of a role, the role does not execute any domain function. In the following sections we will discuss each of these levels of autonomy and possible strategies for implementing such roles in software.

### 5.1. Players with no autonomy

A player with no autonomy is told what actions to execute and always attempts to execute them. In our coffee example, the role provides detailed work instructions that define the process to be followed when executing a role task. The purpose and the system-state of the role are implicit as indicated by the dotted boxes in Fig. 6. The implicit purpose and system-state have been reified into the process instructions by the role designer.

In a static role definition there is no runtime translation from the role's intention to system-state, and from the system-state to process.

However, the performance of the role can vary depending on the execution context provided by the player. This is analogous, in our coffee shop example, to different employees being able to make coffee at different rates. Where the process is defined entirely in the role, as in Fig. 6 above, the player can be viewed as an abstract machine that executes the process provided by the role. The role-player pair acts as a single entity executing in a particular environment. Such environments may have various computational characteristics. In a distributed organization the quality of communication links between roles may also vary. Roles performed by different players (computational contexts) may consequently have different observed performance. In terms of organizational dynamics, changing the role player is changing the machine on which the role is executed. The role identity does not change, nor does its functional relationship to the rest of the system.

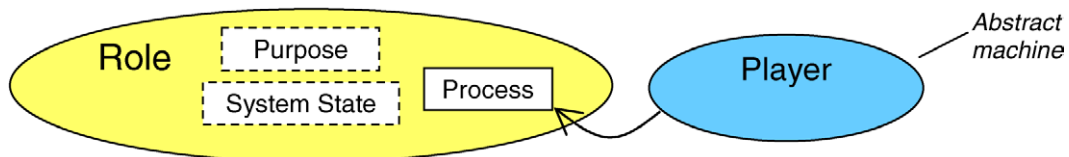


Fig. 6. Alternative 1: Separation between role and player where player has no autonomy.

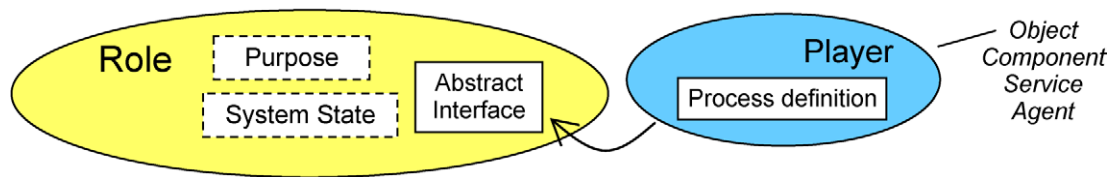


Fig. 7. Alternative 2: Player as execution context and single process.

However, such an implementation strategy does not treat the role as having a separate domain identity to the player. The role itself has to migrate to a new player's execution environment rather than remain in the organization's execution environment. An alternative approach to implementing players that make no process decisions is to have the process interface defined in the role but statically implemented in the player, as illustrated in Fig. 7. This is the approach we have adopted in the ROAD framework (Colman & Han, 2005a). All domain-function is executed in the players. The role is an object that defines a *required* and *provided* interfaces that express the properties defined above in Section 4 (i.e. both functional interactions and their non-functional properties). The role receives/sends messages from/to other associated roles via contracts; buffers incoming messages (if a player is not currently active in the role); and delegates incoming messages to the player.

Other implementation frameworks allow the splitting of domain-function between the role and the player. PowerJava (Baldoni, Boella & van der Torre, 2006a) extends the object-oriented paradigm and Java programming language with a pre-compiler to implement organizational roles. Institutions (like ROAD composites) define roles that are played by players. However, in powerJava, unlike ROAD, roles themselves perform domain-functions and institutions maintain domain state. Institutions give 'powers' to the object playing the roles, rather than having roles statically defined within an institution. Likewise, in Object Teams (Herrmann, 2002; Herrmann, 2005) domain-function can be split between a role and a player (base-object). However, Object Teams does not support adaptivity through indirection of instantiation: once a role-object is created the link to its base-object (player) cannot be changed.

An advantage of having all domain-process defined in the player is that the role structure (organizational composite) remains a pure management abstract. The player can be of any type (object, component, Web service, agent, or back-end of a user interface) as long as the player conforms to the role interface.

## 5.2. Players with process autonomy

A player with process autonomy is given a task to perform in the form of a *system-state*, but it has some autonomy in deciding what steps are executed in order to achieve that task. The player must have the ability to translate a *system-state* provided by the role into a *process* which it can execute. If this system-state is variable then the player may require deliberative capability to effectively perform this translation. On the other hand, translation from the role's *purpose* to *system-state* is implicit – that is, carried out by the programmer when the role is designed.

Where a role provides a system-state to be achieved, rather than a detailed process to be executed, the player must contain the process to achieve the goal. From a viewpoint external to the player (that is, from the role's or organization's perspective) the process is hidden, thus the player has apparent autonomy. If the role is in a stable environment, where the process does not have to change, the player may be implemented as a simple encapsulated object or component that is pre-designed to meet the system-state defined in the role. If the role is subject to environmental perturbation (for example, changing

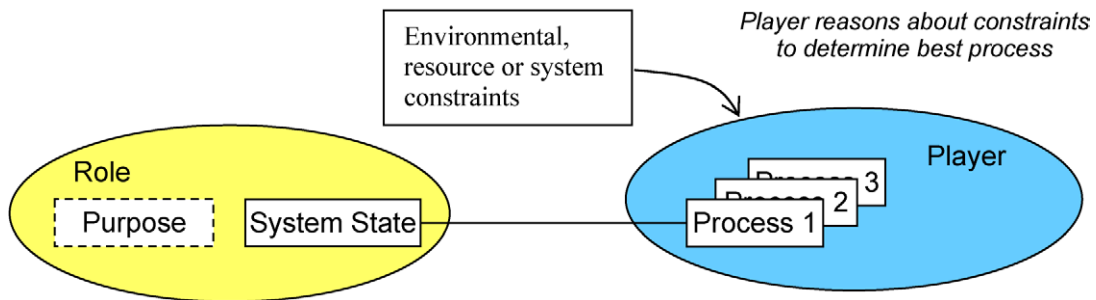


Fig. 8. A player with process autonomy must be able to translate a system-state to a process, create or choose appropriate process given constraints.

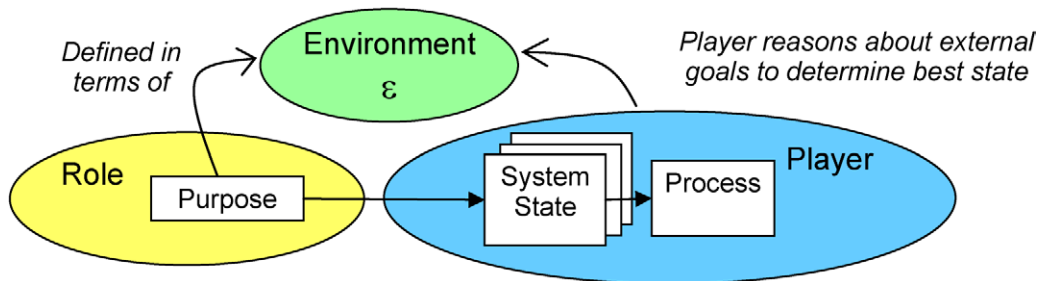


Fig. 9. A player with system-state autonomy must be able to translate a role's purpose into a suitable system-state, and then into a process.

availability of resources) the player may require some deliberative ability to decide what process is the most appropriate one to achieve that system-state as shown in Fig. 8.

A role may also contain pre-defined process plans from which the player selects. These plans may be stored in the role or in the player. Such a player might, for example, be implemented using a BDI agent with a range of plans that can be applied to differing situations and system-states. As with all other types of player, players with process autonomy are performing the role within a computational context that determines the performance of the role-player. Player performance cannot be fully characterized independent of their context, because they are situated entities. However, while actual performance is always related to a situated role-player pair, representations of both the role performance requirements, and the player performance capability, are probably necessary to enable the selection of appropriate players for particular roles.

### 5.3. System-state autonomy

As shown in Fig. 9, a player with system-state is given an external goal which may be satisfied by a number of states. A software example of system-state autonomy would be an operating system that maintains processing capacity by deciding the run-time priority of processes. A number of states could satisfy this goal and the player must choose between them.

At the top-most levels of an organization, players may need to determine the appropriate system-states that best satisfy the role's purpose, given a range of variable internal and environmental constraints. In closed systems, where there is a manageable finite number of system-states (possibly pre-defined states or states defined by a limited set of parameters), it might be possible for a player to have the capability

to evaluate these alternative states and select the one that best matches the role's goal. However, in more open environments, where there are a large number of constraints, it is difficult to automate such capability. Such a role would be typically played by a software developer at design-time, or by a human operator at run-time. These players need the perceptual capability to identify relevant constraints; to devise appropriate system-states; to be able to model the effect of various states on the role's purpose; to determine the best system-state by trading-off costs and benefits; and to devise the processes to realize these states.

Such capability is often required of a human player interacting with the system in a supervisory role, for example, in a mixed-initiative control system. Providing a role interface to the human player allows us to construct systems that have a consistent architecture based on roles, regardless of whether the players are machines or humans. In many control systems, automated control can cope with anticipated perturbations. However, when unanticipated conditions occur, human operators must replace machines as the role players (Rasmussen, Pejtersen & Goodstein, 1994). By abstracting roles from players, systems can be developed that better enable this transition.

#### 5.4. Players with intentional autonomy

The intention or purpose of the role is organizationally defined – it is implicit to the definition of the role. From the organization's viewpoint its players should not have the discretion as to which organizational goal to adopt. A player with intentional<sup>2</sup> autonomy is a free agent with the freedom to decide whether or not to satisfy external goals – it has (or we ascribe to it) its own intentions. If an intentional player is to perform a role in an organization then it has to adopt its role's purposes requirements as its own (*role enactment* in Dastani et al. (2004)'s terminology). However, players may play roles in a number of social networks or organizations (or even multiple roles in the one organization) which can lead to conflicts in priorities and the allocation of resources as shown in Fig. 10.

A *cooperative* entity will fulfill the request if it can. A *competitive* entity only does so if it receives sufficient reward. In the software domain, proactive software agents might exhibit intentional autonomy.

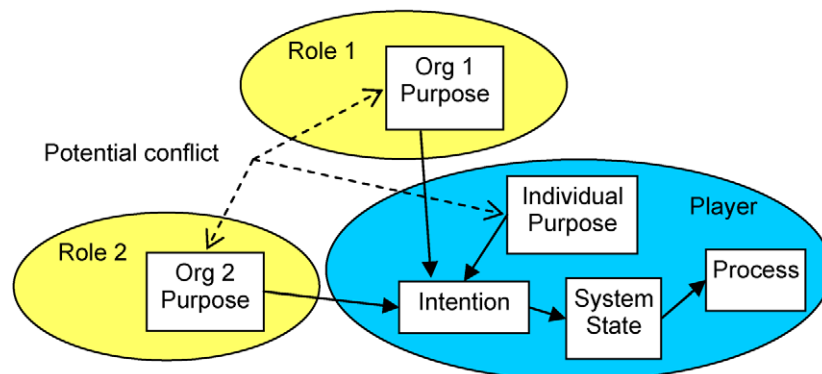


Fig. 10. Organization and individual purpose may have to be resolved by a player with intentional autonomy (a free agent).

<sup>2</sup>We use the word "intention" in the general sense to indicate goal-directed agency as in Dennett (1987) and Searle (1983), rather than in the limited BDI sense (Georgeff et al., 1999) of the selection of a particular course of action. Intention in our usage is more like the "Desire" in BDI.

Cooperative agents attempt to collaborate to achieve system level goals, whereas competitive agents in a market-based system attempt to maximize their own utility.

A player may be required to perform more than one role in an organization, or be a free agent performing roles in a number of organizations. In these cases, the players need some mechanism for prioritizing these conflicting goals, and some way to form an intention to achieve a system-state. Negotiation between the player and the organization about the level of service provision might also be necessary.

### 5.5. Players with constraint autonomy

A player that exhibits constraint autonomy is prepared to violate constraints, norms or even rules to achieve its goals. A greedy software agent may take no account of the computational resources it consumes. A malicious or anti-social agent may deliberately try to harm other agents or the system environment itself. Well run organizations should generally avoid assigning roles to greedy or malicious players. If the use of such players is unavoidable, their behavior has to be tightly controlled. Badly behaved players that can exhibit autonomy from constraints might also be used in organizations provided all their interactions with the organization are controlled. This is the case in the ROAD framework, where all player interaction is via their roles, and all interaction between roles is controlled by contracts. These contracts can be used to ensure the player does not violate organizational constraints. In addition, the contracts can monitor (although not enforce) performance.

### 5.6. Capabilities required of players with different levels of autonomy

Given this conceptual framework we can now define the generalized capabilities that are needed by players to exercise the level of autonomy defined by the organizational role as shown in Table 1.

In (Colman & Han, 2005b) we have argued that to create a viable organizational structure that can achieve system level goals in a complex environment, different role players *must* have varying degrees of autonomy. Players need capability commensurate with the complexity of their respective environments as defined by their role. Mintzberg (1983) has shown that in human organizations, the higher the role's level in the organizational structure, the less formalized and standardized the behavior required of that role. The higher the role is in the hierarchy, the more autonomy and capability that player needs to be

Table 1  
Level of capability needed for players with different level of autonomy

Level of autonomy	General player capability needed
No autonomy	Ability to communicate, follow instructions and effectively use tools and resources provided by the role. The instructions will be 'interpreted' by the player and then executed.
Process autonomy	above + ability to select appropriate processes and tools to complete prescribed tasks.
System-state autonomy	above + ability to sense the environment, to determine which state best fulfills the goal defined by the role in the current environment given the tools and resource available.
Intentional autonomy	Players of roles defined by a <i>closed</i> organization do not have intentional autonomy with respect to their role. The intention of the role is defined by the organization. However, conflict may arise if the player is playing more than one role. In a more <i>open</i> organization (where players may belong to other organizations), conflicts may arise between competing goals. Such a free agent would need the ability to negotiate with the various organizations to which it belongs to try and achieve optimal outcomes.
Constraint autonomy	Players that exhibit constraint autonomy should not be bound to roles in organizations, unless such constraint can be imposed externally on the player by the organization.

able to adapt to environmental perturbations. For example, players with no autonomy cannot be expected to cope with highly variable environments given fixed work instructions. However, autonomy comes at a cost. While a player with system-state autonomy can always perform a highly routinized role, it is not an effective use of resources, particularly if the player has to perform computationally expensive scans of the environment.

## 6. The separation of organizational structure from process

Separating roles from the players that play those roles also allows us to define organizational structures of roles that are separate from the players that perform the function of those roles. In conventional programming structures objects/components/agents talk directly to each other. These mutual references can be hard-coded or can be variable references that are dynamically set. However, such structures are fragile to the extent that, to be well formed, all nodes must be present in the structure. The representation of the structure is also implicit in the references that are embedded, and if object-oriented principles are hidden (if object-oriented principles are followed), in the components themselves. Such an approach also predicates that the entity that is performing a function, must have some representation of the structure in which it will participate; that is, a representation of the context in which the function will be used. This tangles structure and function in the code and inhibits adaptivity. If a component participates in multiple relationships, to change the component requires the restructuring of all those relationships.

An alternative approach has been called the principle of “blind communication” (Oreizy et al., 1999). Lieberherr (1996) similarly proposes that “structure shy” components are a prerequisite for adaptive systems. In these approaches structure is defined separately from the components of that structure, and can be superimposed *a posteriori* on those components. This is the approach adopted in ROAD, as the structure is defined by associating a role with one or more other roles using contracts. On the other hand, a specific role is always bound to a single player. Having a single interface between a role and its player thus simplifies the substitution of players.

A consequence of having players that are structure shy, is that the roles (and the connectors/contracts that bind them) hold a representation of their local structural relationships. A role can be associated with multiple roles but only one player. While all incoming messages (from other roles) are passed to the player, outgoing messages need to be passed to an appropriate associated role. Roles therefore can be thought of as message routers. In Fig. 11 below, Role A passes all incoming messages to its player, but must allocate outgoing messages to either Role B or C, depending on the message type (if B and C

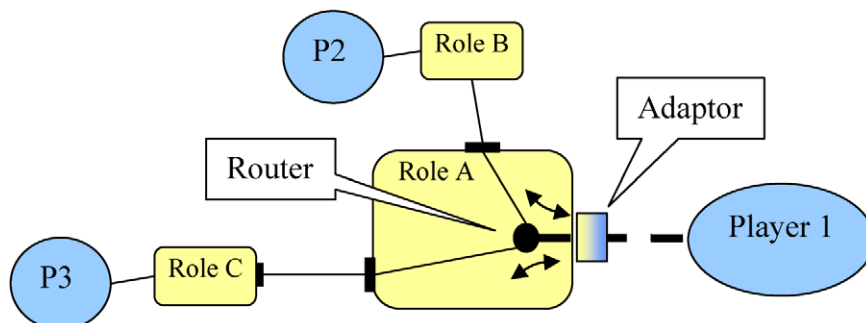


Fig. 11. Role as message router – principle of blind communication of players.

are of different types) or other allocation scheme. Other allocation schemes are needed where structures include multiple roles that fulfill the same basic function (e.g. there is more than one coffee-maker in the organization). For example, if Roles B and C are of the same type (they both can handle the outgoing message), then Role A might route the message to the role-player pair that has the better response time, better reliability or whatever other quality of performance is of interest.

## 7. The preservation of state

Another consequence of the separation of role and players is the need to resolve the question of who is responsible for maintaining state. The maintenance of state is an issue because in dynamic organizations the integrity of the whole needs to be preserved even though the parts that hold state – the roles and the players – may change. We can distinguish two types of state that need to be maintained: communication state and domain state.

### 7.1. Communication state

In an organization, domain-function communication between players is always mediated by their respective roles. Messages to a role may still be generated even though the role is temporarily unassigned or inactive as described in Section 3 above. In order to be viable in the absence of players, organizations need to provide some form of message queuing and storage. The recipient player cannot be responsible for managing and storing these messages because that player may not always exist. A number of alternative approaches are possible to ensure the on-going viability of the organization during the absence or transition of players. These include storing the message in the sending or receiving role (as shown in Fig. 12), or alternatively having the organization store outstanding messages in the contractual associations that connect roles. A further possible alternative of having the sending players hold the message request if the receiver is off-line, is not be a good strategy as the sending player itself may become inactive. It also violates the principle of blind communication as the transmitting player would need to be aware of other roles and whether they have a player assigned.

While the ROAD framework includes message buffering in roles, the approach to how communication state is preserved is perhaps an implementation issue rather than a fundamental attribute of runtime role structures. The scheme employed for message buffering in the role is dependent on the mode of interaction (e.g. push or pull) and type of synchronization used for transactions (e.g. asynchronous transactions). The integrity of message delivery might also be handled in a middleware layer. We address these implementation issues in (Pham, Colman & Han, 2005), and they are not further discussed here.

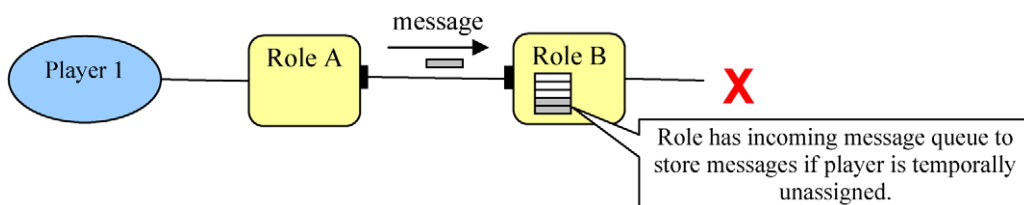


Fig. 12. Roles as messages buffers.

## 7.2. Domain state

The other type of state that needs to be preserved, in the face of changing roles and players, is the state of the process being executed, i.e. the domain state. In object-oriented approaches state is typically encapsulated in the objects. In other approaches state is stored so that it is globally accessible. In the context of a role-oriented organization, a number of alternatives exist as to where the domain state can be maintained. These are:

- State is maintained in player.
- State is maintained in role.
- State is maintained in organization.

The *advantage* of having domain state stored in the player is that it maintains the encapsulation of data and operations on that data (as in object-orientation). As it is always the player that operates on the data, the internal representation can be hidden and decoupled from the system as a whole. This results in the loose coupling of role and player, and facilitates the swapping of player implementation. The player only has to conform to the interface defined by the role.

The *disadvantage* of maintaining state in the player is that, when a player is replaced, any state relevant to the organization must be transferred to the new player. Safe points also need to be defined (e.g. between transactions), when it is permissible to swap players. Alternatively, roll-back or compensation mechanisms would need to be implemented. The problem of how to maintain state during player transfer can be addressed by storing state either in the role or in the organization composite itself. Storing state in the role does not entirely overcome the problem because, in an adaptive organization, roles themselves are created and destroyed. Another approach, as used for example in the powerJava framework (Baldoni, Boella & van der Torre, 2006a), stores state globally in the institution (the organizational composite). Access to such state (and resources) by roles is then controlled by the institution “empowering” roles. Such an approach may be beneficial for storing data related to the composite level of abstraction. However, if it is used to store state that is properly the responsibility of the player, such an approach would break the encapsulation of the player, leading to the well-known problems associated with global data. These problems include inconsistent representation, side effects etc.

As the ROAD framework aims to create adaptive organizations, all domain state is maintained by the players in order to facilitate the swapping of players. As a consequence stateless points in the execution process where it is safe to transition, or alternatively methods for transferring state between players, also needed to be defined.

## 8. Conclusion

Given that highly capable and adaptive humans (compared with software entities) need to form themselves into organizations to achieve complex goals, it seems sensible that software entities in complex environments will need to be similarly organized to achieve system goals (even if they are highly capable and autonomous software agents).

Organization is defined here as the relationships between *roles* in the system, and the processes that maintain the viability of these relationships in response to changing goals and changing environments. An organization-centric view of roles sees roles as nodes in an organizational structure, rather than just behaviors that can be added to an object or agent. In an organization, roles have an identity and existence

that is separate from the players who are assigned them. Roles are first-class runtime entities that can have a number of states with respect to players: assigned or unassigned; active or inactive.

The radical separation of roles and player raises a number of issues that need to be addressed. Organizational structures in complex systems require role-players with various levels of autonomy and capability. The relationship between a role and its player will vary depending on the level of autonomous action required of the player. Developing a software architecture based on the separation of roles from players will facilitate the development of adaptable software systems. A framework that supports such architectures needs to be able to handle bindings between roles and a diverse range of players (objects, services, agents, user interfaces etc.). The framework should be extensible to handle new types of binding. If players are to be easily exchangeable they should be “structure shy”. Consequently, roles need to act as message routers. The framework will also have to handle the problems of preservation of communication and domain state when the organization is restructured. In a ROAD application, roles are stateful interfaces that preserve communication state if there is no player attached. Role-players of various capabilities (including humans in some circumstances) can be dynamically assigned to roles as the demands on the system change, or the environment in which the system operates changes. Such frameworks will be needed in order to build adaptive software organizations, and will need to be based on an ontologically sound conception of role.

## References

- Baldoni, M., Boella, G., & van der Torre, L. (2006a). Bridging agent theory and object orientation: Importing social roles in object oriented languages. In *Proceedings of 3rd International Workshop on Programming Multi-Agent Systems (PROMAS'05) at AAMAS'05*, Utrecht, The Netherlands, July 26, 2005 (pp. 57–75). LNAI 3862, Springer-Verlag.
- Baldoni, M., Boella, G., & van der Torre, L. (2006b). Roles as a coordination construct: Introducing powerJava. In *Proceedings of the 1st International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems (MTCoord'05) at COORD'05* (pp. 9–29). Electronic Notes Theoretical Computer Science, 150(1).
- Baldoni, M., Boella, G., & van der Torre, L. (2005). Introducing ontologically founded roles in object oriented programming: powerJava. In *AAAI Fall Symposium, Roles, an Interdisciplinary Perspective*, Arlington, USA, Nov 4–6 (pp. 5–12). AAAI Press, TR FS-05-08.
- Bäumer, D., Riehle, D., Siberski, W., & Wulf, M. (2000). Role object. In N. Harrison, B. Foote & H. Rohnert (Eds.), *Pattern Languages of Program Design 4*. Addison-Wesley (pp. 15–32).
- Colman, A. (2006). Role-Oriented Adaptive Design. PhD Thesis, Swinburne University of Technology, Melbourne, Australia, <http://www.ict.swin.edu.au/personal/acolman/pub/ColmanROADThesis.pdf>.
- Colman, A. & Han, J. (2005a). Coordination systems in role-based adaptive software. In *Proceedings of the 7th International Conference on Coordination Models and Languages (COORD 2005)*, April 20–23, 2005, Namur, Belgium (pp. 63–78). LNCS 3454, Springer.
- Colman, A. & Han, J. (2005b). On the autonomy of software entities and modes of organization. In *Proceedings of the 1st International Workshop on Coordination and Organisation (CoOrg 2005)*, April 23, 2005, Namur, Belgium.
- Colman, A. & Han, J. (2005c). Operational management contracts for adaptive software organisation. In *Proceedings of the Australian Software Engineering Conference (ASWEC 2005)*, 29 March–1 April, 2005, Brisbane, Australia (pp. 170–179). IEEE Computer Society.
- Colman, A. & Han, J. (2006). Using associations aspects to implement organisational contracts. In *Proceedings of the 1st International Workshop on Coordination and Organisation (CoOrg 2005)* (pp. 37–53). Electronic Notes in Theoretical Computer Science 150(3).
- Colman, A. & Han, J. (2007). Using role-based coordination to achieve software adaptability. *Science of Computer Programming*, 64(2), pp. 223–245, Elsevier.
- Dastani, M., Dignum, V., & Dignum, F. (2003). Role-assignment in open agent societies. In *Proceedings of the Second international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03)*, July 14–18, 2003, Melbourne, Australia (pp. 489–496). ACM Press.
- Dastani, M., van Riemsdijk, B., Hulstijn, J., Dignum, F., & Meyer, J.-J.Ch. (2004). Enacting and deacting roles in agent programming. In J. Odell et al. (Eds.), *Agent-Oriented Software Engineering V, 5th International Workshop, AOSE 2004*, July 19, 2004, New York, USA (pp. 189–204). Lecture Notes in Computer Science 3382, Springer.

- Dennett, D.C. (1987). *The Intentional Stance*. Cambridge, Mass: MIT Press.
- Ferber, J. & Gutknecht, O. (1998). A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceeding of the 3rd International Conference on Multi-Agent Systems (ICMAS 1998)*, 3–7 July, 1998, Paris, France (pp. 128–135). IEEE Computer Society.
- Fowler, M. (1997). Dealing with roles. In *Proceedings of the 4th Annual Conference on the Pattern Languages of Programs*, September 2–5, 1997, Monticello, Illinois, USA. Technical Report #wucs-97-34, Dept. of Computer Science, Washington University.
- Georgeff, M., Pell, B., Pollack, M., Tambe, M., & Wooldridge, M. (1999). The belief–desire–intention model of agency. In *Proceedings of the 5th Inter. Workshop on Intelligent Agents V: Agent Theories, Architectures, and Languages (ATAL-98)* (pp. 1–10). Springer-Verlag.
- Herrmann, S. (2002). Object teams: improving modularity for crosscutting collaborations. In *Net. Object Days 2002*.
- Herrmann, S. (2005). Programming with roles in ObjectTeams/Java. In *AAAI Fall Symposium, Roles, an Interdisciplinary Perspective*, Arlington, USA, Nov 4–6 (pp. 73–80). TR FS-05-08, AAAI Press.
- Juan, T., Pearce, A., & Sterling, L. (2002). ROADMAP: extending the Gaia methodology for complex open systems. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems* (pp. 3–10). Bologna, Italy, ACM.
- Kendall, E.A. (1999a). Role model designs and implementations with aspect-oriented programming. In *Proceedings of the ACM Conference on Object-Oriented Systems, Languages, and Applications (OOPSLA)*, Denver, Colorado, USA (pp. 353–369). ACM Press.
- Kendall, E.A. (1999b). Role modelling for agents system analysis, design and implementation. In *Proceedings of the 1st International Symposium on Agent Systems and Applications*, 3–6 October, 1999, Palm Springs, USA (pp. 204–218). IEEE CS Press.
- Kristensen, B.B. & Osterbye, K. (1996). Roles: Conceptual abstraction theory & practical language issues. *Special Issue of Theory and Practice of Object Systems (TAPOS) on Subjectivity in Object-Oriented Systems* 2(3), pp. 143–160. John Wiley and Sons.
- Kristensen, B.B. (1996). Object-oriented modeling with roles. In *Proceedings of the 2nd International Conference on Object-oriented Information Systems (OOIS'95)*, Dublin, Ireland, Springer-Verlag (pp. 57–71).
- Lee, J.S. & Bae, D.H. (2002). An enhanced role model for alleviating the role-binding anomaly. *Software: Practice and Experience*, 32, 1317–1344. John Wiley and Sons.
- Lieberherr, K.J. (1996). *Adaptive Object-Oriented Software the Demeter Method with Propagation Patterns*. Boston: PWS Pub. Co.
- Loebe, F. (2005). Abstract vs. social roles – A refined top-level ontological analysis. In *AAAI Fall Symposium, Roles, an Interdisciplinary Perspective*, Arlington, USA, Nov 4–6 (pp. 93–100). TR FS-05-08, AAAI Press.
- Mintzberg, H. (1983). *Structure in Fives: Designing Effective Organizations*, Englewood-Cliffs, New Jersey: Prentice Hall.
- Object Management Group (2004). UML 2.0 Superstructure (Final Adopted specification), <http://www.uml.org/#UML2.0>, Last accessed: Oct 2004.
- Odell, J., Nodine, M., & Levy, R. (2005). A metamodel for agents, roles, and groups. *Agent-Oriented Software Engineering (AOSE) V*, Lecture Notes on Computer Science 3382, Springer.
- Odell, J., Parunak, H.V.D., Brueckner, S., & Fleischer, M. (2004). Temporal aspects of dynamic role assignment. *Agent-Oriented Software Engineering (AOSE) IV* (pp. 201–213). Lecture Notes on Computer Science 2935, Springer.
- Odell, J., Parunak, H.V.D., Brueckner, S., & Sauter, J. (2003). Changing roles: dynamic role assignment. *Journal of Object Technology*, 2(5), 77–86. ETH Zurich.
- Oreizy, P., Gorlick, M.M., Taylor, R.N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D.S., & Wolf, A.L. (1999). An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3), 54–62.
- Paesschen, E.V., Meuter, W.D., & D'Hondt, M. (2005). Role modeling in SelfSync with warped hierarchies. In *AAAI Fall Symposium, Roles, an Interdisciplinary Perspective*, Arlington, USA, Nov 4–6 (pp. 149–155). TR FS-05-08, AAAI Press.
- Pham, L.D., Colman, A., & Han, J. (2005). The implementation of message synchronisation, queuing and allocation in the ROAD framework. Technical Report SUT.CeCSES-TR009, Faculty of ICT, Swinburne University of Technology.
- Rasmussen, J., Pejtersen, A.M., & Goodstein, L.P. (1994). *Cognitive Systems Engineering*. New York: Wiley.
- Reenskaug, T. (1996). *Working with Objects: The OOram Software Engineering Method*. Greenwich, Connecticut: Manning Publications Co.
- Searle, J.R. (1983). *Intentionality, an Essay in the Philosophy of Mind*. Cambridge Cambridgeshire, New York: Cambridge University Press.
- Steimann, F. (2000). On the representation of roles in object-oriented and conceptual modelling. *Data and Knowledge Engineering*, 35, 83–106. Elsevier.
- Steimann, F. (2005). The role data model revisited. In *AAAI Fall Symposium, Roles, an Interdisciplinary Perspective*, Arlington, USA, Nov 4–6 (pp. 128–135). TR FS-05-08, AAAI Press.
- Wirfs-Brock, R. & McKean, A. (2002). *Object Design: Roles, Responsibilities, and Collaborations*. Addison Wesley.

- Zambonelli, F., Jennings, N.R., & Wooldridge, M.J. (2000). Organisational abstractions for the analysis and design of multi-agent systems. In *Workshop on Agent-oriented Software Engineering ICSE 2000* (pp. 407–422). Lecture Notes on Computer Science 1957, Springer.
- Zambonelli, F., Jennings, N.R., & Wooldridge, M. (2003). Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3), 317–370. ACM.