Organizational Roles and Players

Alan W Colman and Jun Han

Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne, Victoria, Australia
{acolman,jhan}@it.swin.edu.au

Abstract

This paper discusses the characteristics of organizational roles. We point out differences between the concept of a role as it is commonly used in software development, and the concept of a role as it is implemented in an organization. Roles in organizations have their own identity and do not depend on role players for their existence. We define some characteristic properties of both roles and players in organizational contexts, and show how the functional boundary between a role and its player varies depending on the level of autonomy the player is allowed. Possible implementation strategies for both roles and players are then discussed.

Introduction

In its general usage the concept of a role defines the relationships of an individual within a particular social context. In this paper, we are concerned with software systems in which social contexts are intentionally designed and structured. We call such contexts *organizations*, where organization refers to both the relationship of roles 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.

In this paper our focus is on the binding between the organizational roles and the players that play them. How do we define the boundary between role and role-player? What are the essential properties of roles in software organizations, and how are these properties distinct from the properties of the software entities that play these roles? We refer to the entities that animate roles as "players". 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 and Han, 2005c) 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 include the reasoning capability associated with proactive agents.

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 object or agent-centric. Section 4 examines the essential properties of both roles and players within an organizational structure. 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? We discuss such properties in terms of a conceptual framework based on the autonomy permitted by the role and the capability of the player. Section 5 applies the framework outlined in the previous section to examine appropriate implementation strategies. 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. We conclude with Section 6.

Roles as Design and Implementation Entities

Roles are a recurring concept in both object-oriented and agent-oriented methodologies. 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 and McKean, 2002). In such approaches roles are used in

the modelling and to inform the design, but disappear as entities during implementation.

Other approaches based on role and associative modelling define roles as first-class design and implementation entities (Kendall, 1999; Kristensen and Osterbye, 1996; Lee and 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 (1999) 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 et al. 2005b; Colman and Han, 2005a; Herrmann, 2002). These are discussed in more detail below.

Roles also figure in a number of agent-oriented approaches (Juan et al. 2002; Ferber and Gutknecht, 1998; Odell et al. 2003; Zambonelli et al. 2000). Gaia in particular, (Zambonelli et al. 2003) extends the concept of a role model to an organizational model. 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, give no indication of how these roles are to be realized. 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 code level, we require roles that can be created and manipulated as implementation entities.

Identity and Dependency of Roles and Players

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

- *Visibility*: the visibility and accessibility of the object (a.k.a. player) is restricted by the role
- *Dependency*: a role cannot exist without an object
- *Identity*: An object and its role have the one identity
- *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 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 (1999) and Kristensen (1996) encapsulate roles as implementation entities but allow them no existence separate to 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 identity.

All the above approaches are object-centric (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 and in Baldoni (2005b) and Herrmann (2002), is to look at roles from an organization-centric viewpoint. From this perspective, a role's identity 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 et al. 2005). Roles are the more stable entity within the organizational structure and transient players are attached to these roles.

The organization-centric view of roles accords more with the characteristics of roles in human organizations, such as a bureaucracy or a business. Roles are seen as performing a function within the organization. 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 consist of 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 a single association.

In an organization, each of the *associations* of a functional role can also be characterized. Elsewhere, we (Colman and Han, 2005b) have characterized the control aspects of role associations as *management roles*. As such, management roles are classifiers for the ends of an association, and always come in pairs. We encapsulate these control associations as management contracts. Examples of such roles include supervisor-subordinate, auditor-auditee, predecessor-successor, buyer-seller, and so on. Such *management* contracts 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 how *functional* contracts that bind roles can inherit such patterns of interaction from these abstract *management* contracts.

Roles may be temporarily unassigned to players. A role is a "position" to be filled by a player (Odell et al. 2005). For example, if an employee (player) resigns from their role of 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 the same capability. In a viable organization the role model (organizational structure) is not just a design concept that helps 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 player to a role gives the organization a degree of adaptability in meeting changing goals and environments.

It follows that for roles within an organization, Kristensen's characteristics of *Dependency* and *Identity* do not hold (the other characteristics are still applicable). Roles do not depend on players for their existence and roles have an organizational identity that is independent from their players. Roles therefore can be thought of as having a number of states in relation to the binding with a player. An instance of a role can be either 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).

The separation of organizational roles from the entities that play them, allows the definition of abstract organizational structures that are independent of 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 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 role playing entity, and how to ensure role-player compatibility. We address these issues in the following sections.

The Boundary between Roles and Players

Even though roles within an organization have an existence and identity independent from their players, a player playing a role acts as a unity within the organization. If we separate functional roles from the players who play them, what properties are ascribed to the role and what are the properties of the player?

In general 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 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 and Osterbye, 1996)), and agent-oriented approaches ("agent physical classifier" (Odell et al. 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.

Role definition. The goal of 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 expresso 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 shop manager, peer to the waiting staff and so on. These authority relationships define the valid types of control communication that can pass between actors playing the respective roles.

Player capability requirement. 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.

In general, the organizational role has the following intrinsic 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)
- A definition of the knowledge and skills required in a player to enable performance of the role function. This is an interface definition with which players must be compatible.

A number of other properties are associated with the relationships between a role and its organization (as embodied in other roles). These associational properties could be stored in the respective roles bound in the association, or stored in the association itself. The latter implies that contracts that associate roles are themselves instances that can have state. This is the approach we (Colman and Han, 2005a) have in our ROAD (role-oriented adaptive design) framework. Such associational properties include:

- Communication state. In an organization, 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 defined 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, or alternatively having the organization store outstanding messages in the contractual associations that connect roles. A further 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.

- Performance *requirement* criteria for executing its functions are a property of the relationship of the role with its enclosing organisation, rather than an essential property of the role itself. Such performance criteria are set by the organisation. *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. In our ROAD framework (Colman and Han, 2005a), performance (non-functional requirements) is recorded in contracts that associate roles.
- 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. Such access can be either globally assigned (bad if resources are scarce), or assigned by other roles.

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, as a proxy for its role, with other roles within the organization or with external entities if 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 state to assist in the reasoning process may be required.

The functional boundary between a role and its player may shift 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 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") or 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 contained in 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 hierarchy. 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 operationalisation, 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/instructions 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 it is a relationship between two components at the same level of abstraction. The role is a stateful interface that is implemented by the player.

In (Colman and Han, 2005c) we defined five 'levels' of autonomy based on the constraints to which an entity is subject, namely: no autonomy, process autonomy, system-state autonomy, intentional autonomy and constraint autonomy. We then discussed the relationship between these levels of autonomy and organizational structure. In this paper we will examine in more detail the relationship between autonomy defined by the role (the organizational role can be seen as restricting the player's autonomy) and the capability required of the player to perform that 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). The levels ordered in terms of increasing player autonomy are:

1. No autonomy. The player is told what actions to execute and always attempts to execute them. In our coffee example, the role contains detailed work instructions that define the process to be followed when executing a role task. The purpose and the system-state of the role may be implicit as indicated by the dotted boxes in Figure 1 below. The purpose and system-state is reified into the process instructions by the role designer.

Figure 1. Separation between role and player where player has no autonomy

2. Process autonomy. The player 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.

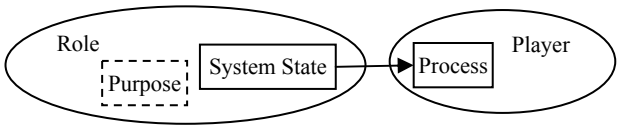

Figure 2. A player with process autonomy must be able to translate a system-state to a process

3. System-state autonomy. The player 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.

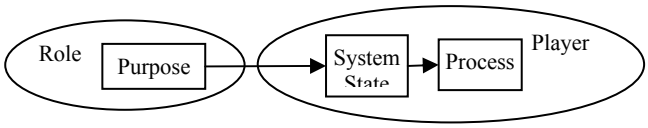

Figure 3. A player with system-state autonomy must be able to translate a role's purpose into a suitable system-state, and then into a process

4. Intentional autonomy. The intention[1] 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 exercise intentional autonomy. They don't have the discretion as to which organizational goal to adopt. A player with intentional 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. 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 Figure 4 below.

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

[1] 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. 2002) of the selection of a particular course of action. Intention in our usage is more like the "Desire" in BDI.

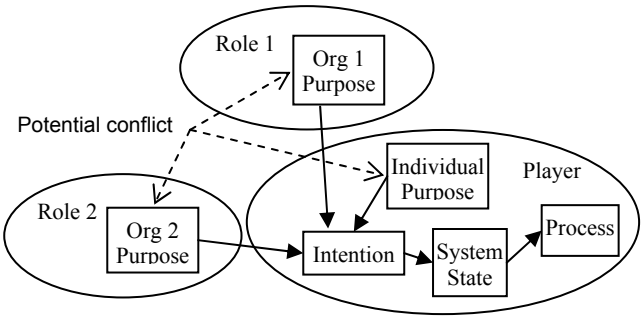

Figure 4.Organization and individual purpose may have to be resolved by a player with intentional autonomy (a free agent)

5. Autonomy from constraints. 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 resource it consumes. A malicious or anti-social agent may deliberately try to harm other agents or the system environment itself. Well run organizations should avoid assigning roles to greedy or malicious players.

Given this conceptual framework we can now define what generalized capabilities are needed by players to exercise the level of autonomy defined by the organizational role.

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

Level of Autonomy	**General 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 then executed.
Process autonomy	above + be able to select appropriate processes and tools to complete prescribed tasks
System-state autonomy	above + be able 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 *closed* 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 *open* 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.

In (Colman and Han, 2005c) we have argued that to create a viable organizational structure that can achieve system level goals, different role players *must* have varying degrees of autonomy if they are to have capability commensurate with the complexity of their respective environments. Mintzberg (1983) has shown that in human organizations, the higher the level a role is 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 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. The next section examines how we might begin to implement roles and players at various levels of autonomy.

Strategies for Implementing Roles and Players

In Table 1 above, we set out some general capabilities required of roles at various levels of autonomy. In this section we use that schema to discuss some implementation options for roles and players.

Implementing Players with No Autonomy

In the case that a player has no autonomy, either the role contains the process definition or the process required by the role is 'hard-coded' into the player. The intention and system-states are implicit in the role. In a static role definition there is no runtime translation from the role's intention to system-state to process. This translation is carried out by the designer of the role rather than the player.

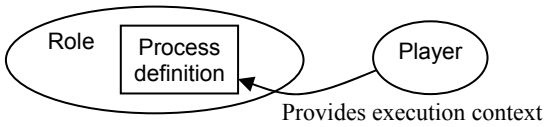

Figure 5. Player as execution context

However, the performance of the role can vary depending on the execution context. 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 Figure 5 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 object 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 change.

However, such an implementation strategy does not treat the role and the player as separate entities. The role itself has to migrate to a different 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 Figure 6 below This is the approach we have adopted in the ROAD framework (Colman and Han, 2005a). All domain-function is executed in the players. The role is an object that defines a required and provided interface. The role receives/sends messages from/to other associated roles via contracts; buffers in-coming messages (if a player is not currently active in the role); and delegates in-coming messages to the player. In ROAD, roles are composed into self-managed composites under the control of an organizer. Power to act within the composite is conferred by its organizer who creates contracts between the roles.

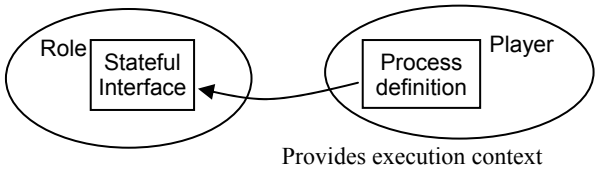

Figure 6. Player as execution context and single process

Other implementations allow the splitting of domain-function between the role and the player. PowerJava (Baldoni et al. 2005a) extends the object-oriented paradigm and Java programming language with a pre-compiler to implement organisational 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) 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.

Implementing Players with Process Autonomy

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. 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 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 in Figure 7 below).

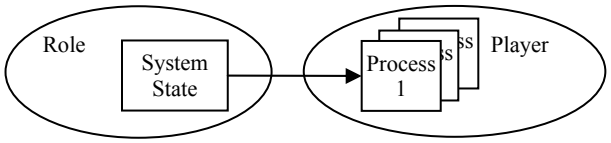

Figure 7. Player chooses appropriate process given environmental constraints

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.

Players with System-state Autonomy

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 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, and Goodstein, 1994). By abstracting roles from players, systems can be developed that better enable this transition.

Players with Intentional and Constraint Autonomy

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 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.

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 ensure the player does not violate organizational constraints. In addition, the contracts can monitor (although not enforce) performance.

Conclusion

Given that highly capable and adaptive humans (compared with software entities) need to form themselves into organizations to achieve complex goals, it would seem 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 independent from the players who are assigned them. Roles can have a number of states with respect to players: assigned or unassigned; active or inactive.

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, agents, user interfaces etc.) ROAD is an example of such a framework. In a ROAD system, role-players of various capabilities (including humans in some circumstances) that can be dynamically assigned to roles as the demands on the system change, or the environment in which it operates changes.

References

Baldoni, M., Boella, G., and van der Torre, L. (2005a) Bridging agent theory and object orientation: Importing social roles in object oriented languages. In *Procs. of PROMAS workshop at AAMAS'05*

Baldoni, M., Boella, G., and van der Torre, L. (2005b) Roles as a coordination construct: Introducing powerJava. In *Procs. of MTCoord'05 workshop at COORD'05,*

Bäumer, D., Riehle, D., Siberski, W. and Wulf, M. (2000) Role Object. In: Harrison et al. (Eds.) *Pattern languages of program design 4*, pp 15-32. Addison-Wesley

Colman, A. and Han, J. (2005a) Coordination Systems in Role-based Adaptive Software. In *Proc. of the 7th Inter. Conf. on Coordination Models and Languages (COORD 2005)* Namur, Belgium, LNCS 3454.

Colman, A. and Han, J. (2005b) Operational management contracts for adaptive software organisation. In *Proc. of the Australian Software Engineering Conf. (ASWEC 2005)* Brisbane, Australia, IEEE.

Colman, A. and Han, J. (2005c) Colman, A. and Han, J., On the autonomy of software entities and modes of organisation *Proc. of the 1st Inter. Workshop on Coordination and Organisation (CoOrg 2005),* Namur, Belgium, 2005.

Dennett, D.C. (1987) *The Intentional Stance*, Cambridge, Mass: MIT Press.

Ferber, J. and Gutknecht, O. (1998) A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems. In *3rd Inter. Conf. on Multi-Agent Systems (ICMAS 1998)* Paris, France, IEEE Computer Society.

Fowler, M. (1997) Dealing with Roles. In *Proc. of the 4th Annual Conf. on the Pattern Languages of Programs* Monticello, Illinois, USA,

Georgeff, M., Pell, B., Pollack, M., Tambe, M., and Wooldridge, M. (2002) The Belief-Desire-Intention Model of Agency, in *Proc. 5th Inter. Workshop on Intelligent Agents: Agent Theories, Architectures, and Languages (ATAL-98).*

Herrmann, S. (2002) Object Teams: Improving Modularity for Crosscutting Collaborations. *Net.Object Days02* Erfurt, Germany

Juan, T., Pearce, A. and Sterling, L. (2002) ROADMAP: extending the Gaia methodology for complex open systems. *Proc. of the First Inter. Joint Conf. on Autonomous Agents and Multiagent Systems, Bologna, Italy, ACM* pp 3-10.

Kendall, E.A. (1999) Role Modelling for Agents System Analysis, Design and Implementation. *First Inter. Symposium on Agent Systems and Applications IEEE CS Press*

Kristensen, B.B. and 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*

Kristensen, B.B. (1996) Object-oriented modeling with roles. In *OOIS'95, Proc. of the 2nd Inter. Conf. on Object-oriented Information Systems* Dublin, Ireland,

Lee, J.S. and Bae, D.H. (2002) An enhanced role model for alleviating the role-binding anomaly. *Software: Practice and Experience* vol 32, pp 1317-1344.

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*

Odell, J., Nodine, M. and Levy, R. (2005) A Metamodel for Agents, Roles, and Groups. *Agent-Oriented Software Engineering (AOSE) V*

Odell, J., Parunak, H.V.D., Breuckner, S. and Fleischer, M. (2004) Temporal Aspects of Dynamic Role Assignment. *Agent-Oriented Software Engineering (AOSE) IV, LNCS 2935, Springer, Berlin*

Odell, J., Parunak, H.V.D., Brueckner, S. and Sauter, J. (2003) Changing Roles: Dynamic Role Assignment. *Journal of Object Technology, ETH Zurich* vol 2, no 5, pp 77-86.

Rasmussen, J., Pejtersen, A.M. and Goodstein, L.P. (1994) *Cognitive systems engineering*, New York: Wiley.

Reenskaug, T. (1996) *Working with Objects: the OOram Software Engineering Method*, Manning Publications Co.

Searle, J.R. (1983) *Intentionality, an essay in the philosophy of mind*, Cambridge, New York: Cambridge University Press.

Steimann, F. (2000) On the representation of roles in object-oriented and conceptual modelling. *Data and Knowledge Engineering* no 35, pp 83-106.

Wirfs-Brock, R. and McKean, A. (2002) *Object Design: Roles, Responsibilities, and Collaborations*, Addison Wesley.

Zambonelli, F., Jennings, N.R., and Wooldridge, M.J. (2000) Organisational Abstractions for the Analysis and Design of Multi-Agent Systems. In *Workshop on Agent-oriented Software Engineering ICSE 2000*

Zambonelli, F., Jennings, N.R. and Wooldridge, M. (2003) Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)* vol 12, no 3, pp 317-370.