JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2003**

Vol. 2, No. 5, September-October 2003

Changing Roles: Dynamic Role Assignment

James Odell, James Odell Associates, Ann Arbor, U.S.A.
H. Van Dyke Parunak, **Sven Brueckner**, **John Sauter**, Altarum Institute, Ann Arbor, U.S.A.

Abstract

Modeling roles as sets of normative behavior that agents can assume has been found to be a useful development technique. An important characteristic of real-world agent systems is that the roles played by an agent may change over time. These changes can be of several different kinds. We describe an illustrative application where such role changes are important, analyze and classify the various kinds of role changes over time that may occur, and show how this analysis is useful in developing a more formal description of the application.

1 INTRODUCTION

The notion of *role* is fundamentally a thespian concept. As humans, we find the perspective and language of the theater a useful analogy for describing and understanding many of the same complex aspects of individual behavior [Odell03]. Since we commonly employ the notion of role in real life for conceptualizing human behavior, it may also serve as a useful device for other kinds of individuals in a MAS—be they life forms, active software constructs, or hardware devices.

In an agent-based system, we define *role* as a class that defines a normative behavioral repertoire of an agent. Roles provide both the building blocks for agent social systems and the requirements by which agents interact.[1] Each agent is linked to other agents by the roles it plays by virtue of the system's functional requirements—which are based on the expectations that the system has of the agent. The static semantics of roles, role formation and configuration, and the dynamic interactions among roles has been examined closely in recent years [Ferbe03] [Caste00] [Dasta03] [Odell01] [Parun01].

[1] Several possible implementation techniques exist for implementing programs that support social entities possessing multiple and changeable class-based roles, including class inheritance and aggregation. In this paper, we will not discuss program-level implementation options for treating role as a class. Instead, our emphasis will by at the analysis-level (i.e., a conceptual and implementation-independent approach).

However, little work has been done on formalizing the temporal aspects of dynamic role assignment. As a result, role modelers refer only informally to actions such as "taking on a role," "playing a role," "changing roles," and "leaving roles." However, such terms can be interpreted ambiguously.

For example, consider the scenarios that follow. These six scenarios are considered to be "changes," yet semantically they all have a different meaning.

1. *Classify* – Add the role of manager to the role of Employee as the result of a promotion.
2. *Declassify* - Remove the role of Manager as the result of a demotion.
3. **Reclassify* - Change from the role of Employee to the role of Unemployed Person.
4. *Activate* - Take up the behaviors of the Manager role as part of the day-to-day business activity.
5. *Suspend* – Stop any Manager behavior and take on just those of the Employee role as part of the day-to-day business activity.
6. **Shift* - Change from an Employee role to a Pet Owner role, where neither is played at the same time as the other. This is a combination of activating one role while suspending another.
   (*These are composite roles, not primitives.)

These scenarios will be discussed in more depth in the following sections. Prior to that discussion, however, some fundamental notions need to be identified and defined.

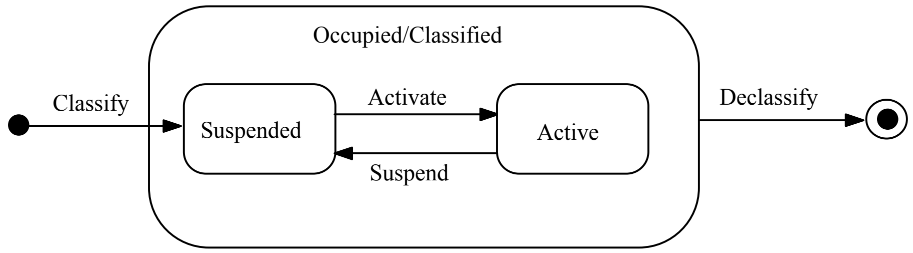


Figure 1 - Statechart depicting some of the permitted states and transitions of an agent in a role.

The statechart in Fig. 1 depicts four of the operations in the scenarios above. Here, an agent comes to *occupy* a role when it is *classified* and ceases to occupy the role when it is *declassified*. Furthermore, while an agent occupies a role, it can be either *active* or *suspended* in playing that role. Transitioning between those two states can be achieved via the *suspend* and *activate* operations. Definitions for these states and transition operations are as follows:

States

Occupied – The state of an entity that is an instance of a particular role.

Suspended - The state of an entity occupying a particular role, when the role has no processes executing.

Active - The state of an entity occupying a particular role that is executing some of all of its processes.

Operations

Classify – An operation that results in an agent being an instance of, or occupying, a particular role.

Declassify – An operation that results in an agent no longer being an instance of, or occupying, a particular role.

Reclassify – An operation that results in an agent being both declassified as one particular role and classified as another.

Activate – An operation that results in an agent entity that is active.

Suspend – An operation that results in an agent entity that is suspended.

Shift – An operation that results in an agent entity becoming both suspended within one occupied role and active in a different occupied role.

These "role-changing" operations can best be discussed by partitioning into categories of dynamics: *dynamic classification* and *dynamic activation*. The sections that follow will describe these notions in more detail. AUML notation will also be proposed.

2 DYNAMIC CLASSIFICATION

Dynamic classification refers to the ability to change the classification of an entity. While dynamic classification applies to both object classes and agent classes [Odell98], in this paper we will discuss it in terms of agent roles. For example in Fig. 2(a), the "Alice" agent changes from being an instance of the roles Employee, Manager, and Salesperson to being an instance of the role Unemployed Person. In other words, with dynamic classification, an agent can be an instance of different roles from moment to moment.

Consistent with [Ferbe03], we insist that each agent have at least one role at all times. Dynamic classification deals with adding additional roles or removing roles beyond the minimum of one. This requirement is analogous with the notion that every human must play the "person" role, whatever other roles they may have. In the case of humans, this minimal role persists throughout the agent's life. It is conceivable that an artificial agent might begin with the minimal role A, add role B, then remove role A, leaving it with the minimal role B. Whether such a fundamental redefinition of the agent is possible will depend on such features as physical equipment associated with the agent and the nature of the platforms on which the agent can run.

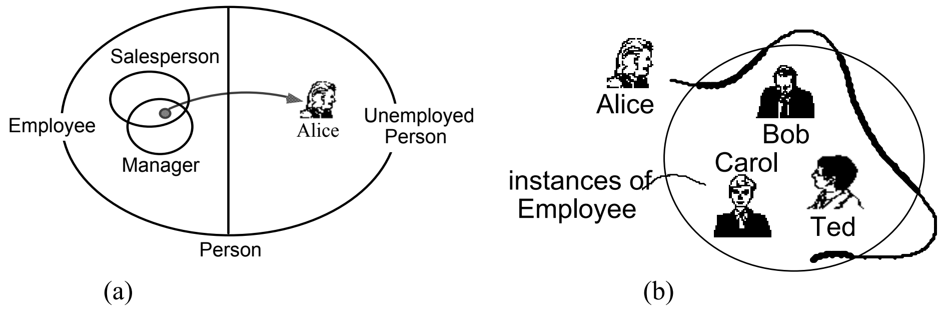


Figure 2 -(a) Dynamic classification refers to the ability to change the classification of an agent's role. (b) The agent "Alice" moving in and out of being classified as an Employee over time.

To become an instance of a given role, the agent is *classified* as an instance of, or *occupies*, that role. Once classified, then, the agent occupies the new role and possesses all of its features. In the opposite process, if an agent is *declassified*, it is removed as an instance of a particular role—and no longer occupies the role nor possesses features unique to that role. Figure 2(b) portrays the "Alice" agent being classified and declassified in terms of the role Employee. At some point in her life, Alice is first classified as an Employee. Later, through some process, Alice is declassified as an Employee: she becomes unemployed. At another point, Alice may become reemployed, followed again by a period of unemployment. This behavior may continue until retirement is reached or the process of death takes place. And where both operations are used at the same time, an agent is said to be *reclassified* when it is both declassified in one role and classified as another.

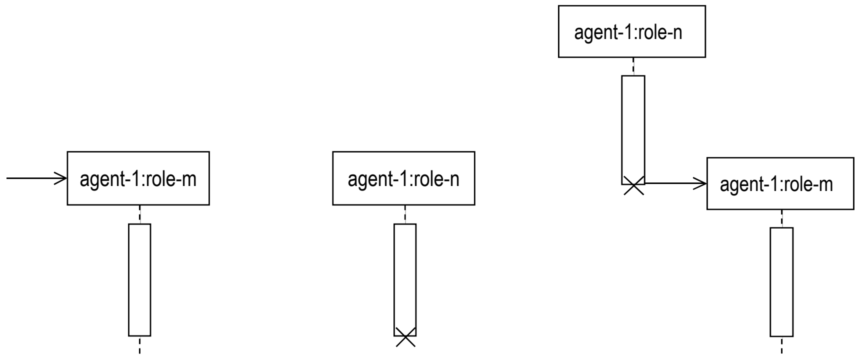
Based on the descriptions above, we can now discuss "change" scenarios 1, 2, and 3 without ambiguity:

1. ***Classify* - Add the role of manager to the role of Employee as the result of a promotion.** In this situation, the person still remains an instance of Employee role. However, the person becomes a new instance of the Manager role. In other words, the person remains classified as an Employee but is now—in addition—classified as a Manager. This is called *classification*.
2. ***Declassify* - Remove the role of Manager as the result of a demotion.** Here, the person still remains an instance of Employee role, but is no longer an instance of the Manager role. *Declassification* has the opposite effect of classification in scenario 1, above.
3. ***Reclassify* - Change from the role of Employee to the role of Unemployed Person.** In this scenario, the person ceases to be an instance of the Employee role and becomes an instance of the Unemployed Person role. In other words, the person was *declassified* as an Employee, while simultaneously being *classified* as an Unemployed Person. This is called *reclassification*.

The table below summarizes the role assignments involving dynamic classification: classification, declassification, and reclassification operations.[2]

Operation	**Pre-state**	**Post-state**
Classify	A and not B	A and B
Declassify	A and B	A and not B
Reclassify	A	B

Both UML and AUML notations would express these three operators as illustrated in Fig. 3. Figure 4(a) indicates agent-1's classification as role-*n*; in Fig. 3(b), agent-1 is declassified as role-*m*; in Fig. 3(c), agent-1 is being reclassified from role-*m* to role-*n*.



(a) Classification (b) Declassification(c) Reclassification

Figure 3 –AUML notation for classification, declassification, and reclassification.

3 DYNAMIC ACTIVATION

In the previous section, we have seen how "Alice" may be classified and declassified as Employee and Unemployed Person roles over time. Orthogonally, she may get married and become an instance of the role Married Person and Wife. Additionally, she may be confirmed as a Supreme Court Justice or buy a pet and become a Pet Owner. While she may be a Supreme Court Justice for life, she may later give the pet away and be removed from the Pet Owner set. Every agent, then, must always be classified in at least one role — where Agent can be thought of as a role in its own right.[3]

In her lifetime, Alice may be an instance of many roles. This means, first, that the roles that apply to an entity can change over time (dynamic classification). Second, it means that an entity can have multiple roles that apply to it at any one moment. When an

[2] For completeness, arguably two more operators could be added: *create* and *terminate*. The create operator classifies entities that did not previously exist. Here, the prestate for an entity would be null, and the poststate a particular role. Termination is the opposite, where the prestate would be for an entity in some role and the poststate would be null.

[3] Having Agent as a role is a controversial point. However, in [Odell, 2003] we defined *role* as a class that defines a normative behavioral repertoire of an agent. The basic class called Agent defines the normative behavioral repertoire for agenthood.

entity is an instance of more than one role this is called *multiple classification* (not to be confused with multiple inheritance).

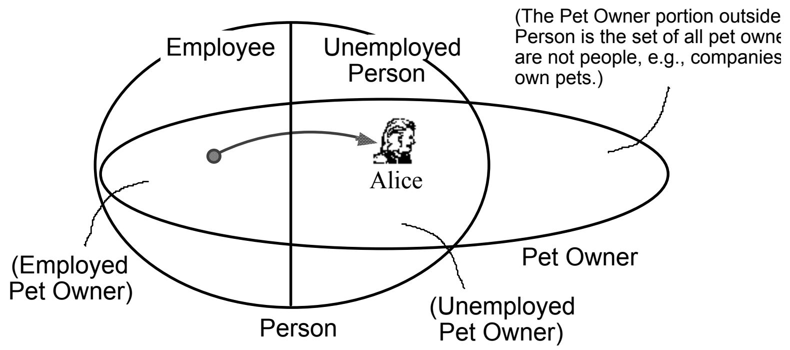

Employee
Unemployed Person
(The Pet Owner portion outside Person is the set of all pet owne are not people, e.g., companies own pets.)
Alice
(Employed Pet Owner)
Pet Owner
Person
(Unemployed Pet Owner)

Figure 4 - “Alice” involved in both dynamic classification and multiple classification.

Multiple classification is a useful notion here, because in a society-based system an agent is likely to be active (or quiescent) in multiples roles at the same time. However, what does it mean for a role to be “active”? In some sense, a quiescent agent that is waiting for a message or some signal to awaken could be considered active in its role, because alertness can be thought of an activity. UML 2.0 [OMG, 2003] offers a useful refinement by distinguishing between user-defined actions (which are represented explicitly in sequence diagrams and activity diagrams) and fundamental system actions such as I/O, invocation, and data flow (which are not represented as actions in these diagrams). In UML, each activation, or *execution occurrence*, has some duration and is bounded by a start and stop point. We propose to take advantage of this refinement in the following unification:

- *We adopt the UML 2.0 definition of action.* Any unit of behavior that has started and has not yet ended is considered “active”. Otherwise it is “inactive”.
- *We use the basic role of* AgentId *to specify primitive behavior.* (Id in AgentId is meant to suggest the Freudian sense of primal basic urges, not the sense of “Identity.”) Behavior such as controlling, handling data flows, and waiting for messages and signals can be thought of as “primitive” actions that all entities must possess to be agents. Therefore, any entity playing the role of AgentId can exhibit this basic behavior, deferring “higher-level” behavior to user-specified actions in more specialized roles. Furthermore, these basic behaviors are themselves actions. For example, actions that support listening for messages and signals, *by definition,* begin the moment an entity is classified an AgentId and cease when the entity is no longer an AgentId. This default role satisfies the criterion by Ferber and Gutknecht [Ferbe03] stating that “every agent plays at least one role,” in this case the AgentId role.
- *We consider roles other than AgentId to be active only when their user-defined actions are active.* Activity of primitive actions is attributed to the concurrently executing AgentId role, not to the user-specified role.

Dynamic activation involves the following operations: activate, suspend, and shift. These operations are discussed in "change" scenarios 4 through 6, as follows:

4. ***Activate* - Assume the behaviors of the Manager role as part of the day-to-day business activity.** Here, the person still remains an instance of both Employee and Manager roles. However, when dealing with the employee's boss, the person is *active* in playing Employee role. Yet, when the employee starts addressing her subordinates, she also becomes active in the Manager role. In other words, the person begins the scenario as active only in the Employee role, and ends being active in both the Employee and Manager roles.
5. ***Suspend* - Change from the role of Manager to the role of Employee as part of the day-to-day business activity.** As with the previous example, the person still remains an instance of both Employee and Manager roles. In this scenario, the person might start the day as a Manager role by approving an expense report for a subordinate. However, if the person then reports to her boss, she *suspends* her role as Manager. In other words, the person begins the scenario as classified and active in both Employee and Manager roles, and ends with the Manager role suspended while the Employee role remains active. (This is the opposite effect of the scenario 3, above.)
6. ***Shift* - Change from an Employee role to a Pet Owner role, where neither is played at the same time as the other.** Here, the person still remains an instance of both Employee and Pet Owner roles. However, this differs from scenarios 4 and 5, because the person is not active in both roles at the same time. Here, the person suspends his Employee role and becomes active in the Pet Owner role—yet consistently remains classified in both Employee and Pet Owner roles.

The table below summarizes the role assignments involving activation-related classification: activate, suspend, and shift.[4]

	Prestate			**Poststate**	
Operation	**Active**	**Suspended**		**Active**	**Suspended**
Activate	A	B		A and B	
Suspend	A and B			A	B
Shift	A	B		B	A

Both UML and AUML notations would express these three operators are illustrated in Fig. 5. Figure 5(a) indicates agent-1 is activated as role-m; in Fig. 5(b), agent-1 is suspended as role-m. In Figs. 5(c) and 5(d), the role of agent-1 shifts from role-n to role-m. Asynchronous messages (indicated by the open arrowhead) do not require a response. Therefore, the shift would proceed from the end of an execution occurrence bar (the thin

[4] Four combinations were omitted from this table. The situations where the pre- and poststates have only suspended roles and where the pre- and poststates have only active roles is not interesting here because there are no state changes. The prestate, where only active role(s) exist and become only suspended ones, is just two concurrent cases of suspension. The prestate, where only suspended role(s) exist and become only active ones, is just two concurrent cases of activation.

triangle over the lifeline) for one role to the beginning of the execution occurrence bar for another (Fig. 5(c). In contrast, synchronous messages (indicated by the solid arrowhead) require a response before proceeding. Figure 5(d) depicts an agent in role-n sending a message that activates role-m. Since the message is synchronous, all role-n processing is suspended until control is returned from role-m (the dashed arrow).

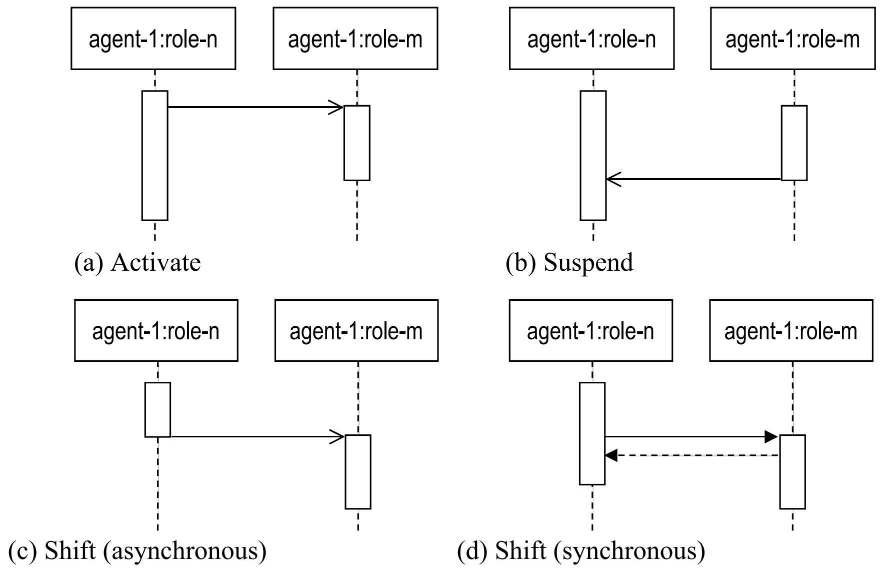


Figure 5 – UML and AUML notation for expressing activate, suspend, and shift. For graphical clarity, message lines can be supplemented with stereotypes. For the activate, suspend, and shift operations, the stereotypes would be «activates», «suspends», and «shifts».

4 CONCLUSION

Roles are increasingly recognized as a valuable abstraction for modeling groups of agents. In dynamic environments, an agent may change the roles it plays over time. Analysis of these changes show that they fall into two general categories.

1. The more conventional concept is Dynamic Activation. An agent may incorporate multiple roles but not be active in all of them at the same time. Varieties of Dynamic Activation describe the different patterns in how an agent activates or suspends the various roles that it possesses.
2. Dynamic Classification deals with the more fundamental binding between an agent and a role. Straightforward mechanisms for role assignment in contemporary programming languages (e.g., inheritance from a class defining the role's behaviors) are static and persist for the agent's lifetime. However, the concept of Dynamic Classification encourages us to conceive of roles being bound to an agent after the agent is instantiated, and unbound without terminating the agent.

The notion of roles invites a new approach to agent programming, in which the unit of agent invocation is the role rather than the individual behavior. In such a role-oriented programming environment (ROPE), invocation consists of passing an agent a role and an execution environment (compare the notion of Agent Coordination Context in [Omici02]), and it is up to the agent to carry out the role in that context. Development of ROPE is a future opportunity for this line of research.

5 ACKNOWLEDGEMENTS

This work is supported in part by DARPA, contract F30602-02-C-0196 to Altarum, under DARPA PM Vijay Raghavan. The views and conclusions in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

The authors are grateful to their colleagues Paul Chiusano, Bob Matthews, Mike Samples, and Peter Weinstein for insightful review and comments on earlier drafts of this paper.

REFERENCES

[Caste00] Castelfranchi, Cristiano. “Engineering Social Order”, *Engineering Societies in the Agent World*, Springer, Berlin, pp. 1-18.

[Dasta03] Dastani, M., V. Dignum, and F. Dignum. “Role Assignment in Open Agent Societies” in *Proceedings of AAMAS’03, Second International Joint Conference on Autonomous Agents and Multi-agent Systems*. 2003. Melbourne, Australia.

[Ferbe03] Ferber, J., O. Gutknecht, et al. (2003). “Agent/Group/Roles: Simulating with Organizations.” *Fourth International Workshop on Agent-Based Simulation (ABS03)*, Montpellier, France.

[Marti98] Martin, J. and J.J. Odell, *Object-Oriented Methods: A Foundation*. UML ed. 1998, Englewood Cliffs, NJ: Prentice Hall.

[Odell98] Odell, J.J., *Advanced Object-Oriented Analysis & Design using UML*. 1998, Cambridge, UK: Cambridge University Press.

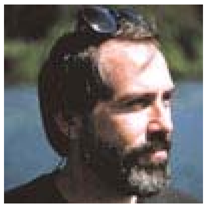
[Odell01] Odell, J., H.V.D. Parunak, and B. Bauer. “Representing Agent Interaction Protocols in UML,” in *Agent-Oriented Software Engineering*, P. Ciancarini and M. Wooldridge, eds. 2001, Springer: Berlin. p. 121-140.

[Odell03] Odell, J., H.V.D. Parunak, and M. Fleischer. “The Role of Roles in Designing Effective Agent Organizations,” in *Software Engineering for Large-Scale Multi-Agent Systems*, A.F. Garcia, et al., eds. 2003, Springer-Verlag: Berlin.

[Omici02] Omicini, A. “Towards a notion of agent coordination context”. In D. Marinescu and C. Lee, Editors, *Process Coordination and Ubiquitous Computing*, 187–200. CRC Press, 2002.

[Parun01] Parunak, H. Van Dyke and James Odell (2001) “Representing Social Structure using UML”, *Proc. of the Agent-Oriented Software Engineering Workshop, Agents 2001 Conference*, Paolo Ciancarini, Michael Wooldridge, and Gerhard Weiss eds., Agents 2001 conference, Montreal, Canada, Springer.

About the authors



James J. Odell is a consultant, writer, and educator in the areas of object-oriented and agent-based systems, business reengineering, and complex adaptive systems. He has written four books on object orientation and has two books in progress on agent-based system design. His website is http://www.jamesodell.com.

Dr. Van Parunak is Altarum's Chief Scientist. He is currently working on the applications of complex adaptive systems, with special emphasis on fine-grained agent software architectures for modeling, control, and collaboration. He has written numerous seminal papers in this area, which are available at http://www.erim.org/~vparunak.

Dr. Sven A. Brueckner is a member of the Technical Staff at Altarum. His website is http://www.erim.org/~sbrueckner.

John A. Sauter is the Group Leader for Emerging Markets in the Enterprise Solutions Division of Altarum. John has over 20 years[1] experience in systems design using agent-based methods for simulation, modeling, autonomous vehicle control, automotive logistics, plant floor scheduling and control, flexible manufacturing, communications systems, embedded systems, material transport systems, and automotive test equipment. He holds a BS in Chemistry and Cellular Biology from the University of Michigan (1976).