

An Argument for the Elimination of Roles

Dennis Heimburger

Dept. of Computer Science
University of Colorado
Boulder, CO

Leon Osterweil

Dept. of Computer Science
University of Massachusetts
Amherst, MA

Abstract

The Role approach currently used in many process models is seriously flawed. It imposes rigid boundaries separating people from the actions they might perform. It also combines together a number of concepts that are best kept separate. We propose an alternative to Roles based on loosening boundaries and on decomposing the Role concept into its constituent elements.

The Role Model In Process Systems

A common approach for representing people in a process formalism is to use the concept of *Roles*. A Role typically represents a sort of “user type” representing some organizational concept such as manager, process programmer, or product programmer. Associated with a Role is the right to perform selected operations on selected objects in the process system. Thus, various tasks (such as coding of specific modules) would be associated with a particular Role such as product programmer. A real person Sue, say, would carry out the coding of some module by becoming associated with the product programmer Role¹.

The Role approach would appear to have some difficulties in practice due to the rather rigid boundaries that it imposes on peoples actions. In particular, these boundaries cause two complementary problems: *inappropriate aggregation* and *Role proliferation*. Inappropriate aggregation means that a particular Role may accrete access to a large number of objects, This happens because people, in the guise of particular Roles, tend to discover new objects for which access is required in that Role. In any given use of the Role, a real person will have access to additional, unintended objects not needed to complete any particular action.

In order to avoid this kind of aggregation, Roles may be created, each of which has a more precise list of accessible objects; but this leads to the proliferation of large numbers of Roles each with access to a small number of objects. If, as is often the case, a person can only act in one Role at a time, this can lead to a form of thrashing in which a person continually switches Roles in order to accomplish anything.

¹ Actually, the combination is a Role instance. In this paper, we will not be precise about the distinction between the Role (type) and the Role instance.

De-constructing Roles

These problems occur with the Role concept because it combines several elements that should in fact be kept as independent as possible. Specifically, Roles combine the concepts of *user types* and *access rights*.

The Role aggregates together sets of people that are doing similar activities. Thus, the Programmer Role serves to aggregate together all the people who are acting as a programmer with respect to the process system. In effect, then, a Role is a *type* of user of the process system.

By itself, the user type defined by a Role has no necessary consequences. But Roles also serve as entities to which access rights are attached. Typically, this takes the form of specifying a set of objects to which the Role has the right to invoke a specified set of operations. Thus a Role acts as a form of *capability domain* combining a subject (user) with access rights to specified objects.

It is this combination of access rights with user types that causes the rigidity of Roles. An active agent (typically a person) in a process system generally cannot act except through a Role to which it has been bound (to create a Role instance).

An Alternative to Roles

We propose a replacement for Roles based on decomposition of the Role concept. This entails giving the concepts of *user type* and *access rights*, explicit representations. There is one complication in doing this decomposition. In order to talk about access rights independent of the user type, it is necessary to change from a capability model to an *access list* (ACL) model.

Protection Models

In order to understand how protection is represented in Roles, We want to resurrect an old, simple, and useful model for access protection: *access matrices*. In an access matrix, protection is modeled in terms of *objects*, which label the columns, and which are the various things that can be accessed, The rows are labeled by *subjects*, which are the things that do the accessing. It is normally assumed that the set of subjects is a subset of the set of objects. At the intersection of object O_i and subject S_j , we set the operations $E(i,j)$ that the subject may invoke on the object. A blank entry indicates no access.

Traditionally, there are two ways to “store” this access matrix: by rows and by columns. Storing by columns means that each subject has associated with it the set of objects that it can access and the operations it can invoke on those objects. This is usually called a *capability* representation. Storing by rows means that each object has associated with it the set of subjects that it can access and the operations each subject can invoke on the object. This is usually called an *access control list* (ACL) representation.

Role Access Models

For its protection model, we claim that a Role uses a capability model. That is, a Role is a subject and attached to it are the capabilities (objects X operations) it has for accessing various process-related objects. In order to separate Roles into their constituent pieces (types and rights), we propose the replacement of the capability model of Roles with an ACL model.

Consequences

The obvious question is: are there any practical consequences to decomposing Roles (i.e., into user types and access lists). The answer appears to be yes. As an example, consider what is necessary in order to give over control of a coding task from a specific programmer to a specific documenter. With traditional Roles, one is required to either 1) give the documenter the roles of the programmer, or 2) create a new role (programmer-documenter?) with the appropriate capabilities, or (3) add access to the coding task to the documenter Role.

Case one has the potential for giving the documenter access to too much information, because the existing roles may have capabilities to objects other than the specified task. Case two works, but requires the creation of a whole new Role, and this leads to the proliferation of Roles. Case three also works, but gives all documenters access to the task rather than restricting it to specific documenter. By contrast, the ACL model can achieve the desired effect more precisely by simply adding the specific documenter to the access control list for the task object.

Conclusion

The Role approach currently used in many process models is seriously flawed. It combines together a number of concepts that are best kept separate. We propose an alternative to Roles based on decomposing the Role concept into its constituent elements and using access control lists instead of the capabilities used by Roles.

Acknowledgements

This material is based upon work sponsored by the Air Force Materiel Command, Rome Laboratory, and the Advanced Research Projects Agency under Contract Number F30602-94-C-0253. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.