Salient Features of DMMA-based ObjectRoleEvolution

- We are endeavouring to model evolving objects in class-based object oriented systems and OODBMS. We are using multiple abstractions for the same.
- Our ObjectRoleModel consists of a non-evolutionary part and an evolutionary part that consists of variable number of abstractions.
- Databases store objects over long periods and the represented entities evolve during the time. Our work dwells on extending the class-based OOSs to handle evolving objects well where objects play roles. At a point of time, an entity is represented by the entity object itself and the current active role instance set. To play a role, the entity has to acquire relevant role type and role instance.

- As a result of object role dynamics, the entity object's type, state and behaviour experience transitions across the complete role set that it can assume in its life time.
- This is a set-oriented approach as opposed to earlier works in the area that use object hierarchies, class hierarchies, root hierarchies etc. The multiple abstractions capture multiple roles and facilitate clear-cut separation of concerns and a powerful method resolution. There is no need for inheritance to capture objects with roles.
- We extend the C++ language with multiple abstractions for objects with roles. The object middleware such as COM already have provision for multiple abstractions and we want to explore the object role modeling issues in COM.

Operational Semantics for ObjectRoleDynamics

- **activate()** for the root object to acquire a role abstraction. Only then can a role be instatiated. The same operation is used to create a role instance.
- **deactivate()** to delete a role abstraction from the root object once for all. The same operation is used to delete a role instance as well.
- **inactivate()** to suspend a currently active role abstraction. The same operation is used to suspend a role instance. So there are active role abstraction/instance sets and inactive role abstraction/instance sets associated with the root object.
- **reactivate()** to reactivate() an inactive role abstraction/instance

- The role type set and role instance set will ever consist of three parts the active set, the inactive set and the nonactive set.

Conclusions

- A new ObjectRoleModel is presented. Multiple abstractions approach. No class hierarchies, no root hierarchies, no inheritance at class level or object level.
- Object evolution through class evolution and role type evolution.
- Powerful method resolution
- Better role modeling capability
- Multiple abstractions approach. Under exploration: Is it well suited to capture distributed roles? In a distributed system, several roles of an entity may be stored at different sites. Each role uses a different object identity (one per role).
- We want to study the ObjectEvolution in general and ObjectRoleEvolution in particular in a complete way. We want to capture the object evolution using class evolution, object evolution and role evolution.

- We want to extend the work towards role patterns studying RolesPlayed pattern of Peter Coad and RolePatterns work of Kristensen to start with.
- We want to extend the work to the sender/receiver based applications where message passing occurs between objects where each one plays different roles, bearing varying role sets. The idea is how to efficiently capture the message polymorphism in such an environment. Some issues are Sender-based/Sender's class based/Sender's context-based method interpretations/executions.