DESIGN OF AN EXTENSIBLE SYSTEM SUPPORTING PERSISTENT OBJECTS

Although software has become the most important component of computer systems, software production is suffering from a cronic crisis of unproductivity. Many different approaches including the total quality management and integrated software development environments are considered in response to this problem.

A long standing issue in operating system design has been the notion of generality. While general purpose systems can run quite different type of applications, in many cases these systems are not suitable for certain types of operations. While extending the system through programming is possible, it is usually not a very trivial task.

The primary goal of this research is to define an extensible and tailorable computing system model which will attack the programmer productivity issue from the technical side. The resulting system is suitable to be used as a base for an extensible system for different types of application domains.

Extensible Persistent System (EPS) suggests a new model for extensible systems based on a unifying view of persistency. Data and processes are viewed as persistent objects, and handled through a uniform interface. The system definition can be developed recursively using the notions of active, and passive objects. We believe this simple yet powerful view of the system modeling will support application developers and system programmers in understanding and extending the system. Moreover, the resulting extended system will not need extra system administration tasks and complex configuration management. Despite of its revolutionary nature, EPS is easy to learn and adapt since it is built on UNIX operating system.

Persistent objects are objects whose life span are longer than their creator programs, therefore provide a better abstraction for modeling the real world by computer programs, and save programmers from writing code for supplying this model manually. Passive objects, one of the two basic abstractions of EPS, are persistent data only objects. Active object, on the other hand include not only data but also methods and act as servers. Active objects are servers that provide extensibility capability to EPS, they provide system services at user level , they provide system services as user level services, quite similar to other modern operating systems. EPS, provides an comprehensive support for the development and management of active objects.

An open issue for security architectures based on capabilities was the cancellation of once distributed capabilities. While some methods of complete cancellation is proposed, selective cancellation which means to remove access rights of some processes while leaving others valid was a problem in many systems. Another problem in access rights cancellation is the restriction of some rights while leaving others. EPS, lets a capability owner to remove a capability completely, or partially or restrict the once given access rights selectively. In EPS we also tried to apply a unique solution for transfer of capabilities from one process to another.

In a conventional system, variable names and scopes are checked by the compiler according to the language rules, which resolves name conflict problem in the compile phase. Also, since

the variable names are visible to the programmer, there is little chance of making mistakes or losing a variable. Another problem with a persistent system arises because of the long life span of persistent objects and their wide scope in contradiction to conventional systems. A persistent object identifier, which is expected to be unique, may have already been used before by another programmer. The process of finding a unique identifier can easily become a tedious job, even impossible. This issue is attacked by a naming structure consisting from two parts; owner object name, and persistent object name. The name resolution algorithm based on this naming structure is a flexible one giving the programmer highest freedom possible without jeopardizing the object safety.

EPS also suggests a flexible, programmer controlled approach to synchronization and deadlock prevention. The synchronization approach is similar to that of relational database management systems, which seems to be a successful solution used widely in the industry. For the deadlock prevention, a choice of gradually or one-step lock release is supplied to the programmer which shall be flexible enough to satisfy different types of application development needs. A distributed transaction mechanism is also provided, that can be a basis for further research.

EPS is implemented by following components; inter-object communication service (IOC), client object library (COL), active object library (AOL), naming and protection server (NPS), object server (OBS), and a preprocessor on Linux operating system.