Issues in Designing Object Management Systems

Geoff Clow[1] *and Erhard Ploedereder*[2]

[1] Softech Incorporation
16875 W Bernardo Drive #201
San Diego
CA 92127
USA

[2] Tartan Laboratories Incorporated
300 Oxford Drive
Monroeville
PA 15146
USA

Abstract

This paper summarizes the discussions of the Object Management System (OMS) session at the Chinon Workshop. The session identified numerous capabilities which might be required in an OMS. The facilities which were agreed upon as essential to an OMS are presented in Section 1, OMS Core Facilities. A number of issues in the realization of these and other capabilities, influenced in part by specific application scenarios, are discussed in Section 2, OMS Requirements Issues. Promising applications requiring further investigation can be found in Section 3, Perceptions for the Future. Some global observations on the past and future conduct of the OMS field are summarized in Section 4, Concluding Observations.

1 OMS Core Facilities

An object management system (OMS) provides services for the management of project data. Its crucial and generally agreed upon capabilities are described in this section.

1.1 Capturing Data

Project data includes life-cycle products (*e.g.*, plans, designs, reports, code, documentation) and process information (descriptions of development practices, models, and states). Effective project management presupposes capturing information about the development process and the products of that process. Data capture is a service provided by the OMS to project support tools. However, this service can go beyond merely providing a data repository, if the OMS actively captures information rather than just storing the data explicitly provided to it.

Automating the capture of and response to data is an on-going research topic crossing the domains of object and process management, as discussed in Section 3 below.

1.2 Data Modelling

The conceptual structures used in presenting data stored in the OMS significantly affect the utility of the OMS. The goal of data modelling is to represent data in a way which exposes its structure. One measure of the quality of an OMS is the granularity of the information it captures about the data it stores. Stream I/O to file contents is an example of the lack of data modelling by the OMS, as all data interpretation is left to the programs which read the data. Conversely, one frequent example of an OMS data model is a typed Entity-Relationship-Attribute (ERA) representation, in which the OMS specifies to some extent the structure and possible values and operations for the data. The particular model in which an OMS should present data (*e.g.*, ERA, Object Oriented, Relational), and whether there should be one or more data models within a given OMS, remain issues for experimentation.

Data modelling also facilitates other OMS services, such as data sharing and data integrity assurance.

1.3 Data Sharing

Project Support Environments (PSEs) include many diverse tools operating on the same data. Ideally, such common data should not be duplicated in multiple data models and representations or redundantly derived. In today's practice, however, we find that such duplication occurs (with its ensuing integration and consistency problems), as many tools implement their own hidden OMS or utilize multiple OMSs with distinct interfaces and data models. Integration of tools by utilization of a common set of interfaces to a single OMS would facilitate data sharing but makes migration of existing tools into a more integrated PSE (IPSE) considerably more expensive. A further alternative to a single, encompassing OMS or multiple, independent OMSs is a single OMS, some of whose administered objects are in turn object bases (nested OMSs); this provides for sharing of "coarse-grained" data while permitting alternative data modelling of "fine-grained" data.

1.4 Data Integrity

Sharing data implies a need to agree on the legal operations on that data and on their sequencing or concurrency. That agreement can be provided primarily through the OMS, without the need for direct coordination of tool authors. Access synchronization and transaction mechanisms can coordinate the activities of multiple tools and can help to maintain the consistency of multiple objects. Typing facilities control the values of data and the primitive operations applied to data. Access controls determine the processes which may be applied to data

by a given user. History, logging and trigger mechanisms have applications in determining or maintaining data integrity. Process management can be integrated with object management to control the complex operations applied to data. The number and sophistication of these facilities is partially a precondition for integrating diverse tools, users and data, and partially the means to capitalize on such integration. The OMS plays a crucial role in enabling the generation of such facilities.

1.5 Secrecy

Sharing of data (and information about such data) in an integrated environment must be managed and monitored to control the dissemination of sensitive information. An OMS needs to provide suitable mechanisms for discretionary and mandatory access control.

2 OMS Requirements Issues

There is wide agreement on the need for the cited core facilities of an OMS. However, there remains substantial debate on how best to achieve those capabilities (*e.g.*, one OMS, multiple OMSs, nested OMSs) and on certain of their requirements.

Many critical OMS requirements issues appear to be subsumed in a single general question. The clients of an OMS are tools (requiring project data management services). The question is whether all tools should utilize the same OMS (the same interface providing access to the same object base), or different tools should utilize different OMSs? A single OMS is one means to integrate tools and users: All project data are available to all tools (within appropriate integrity and security limits), and a uniform data model may be presented to tools and users. Tools that are generic to many life-cycle activities, such as configuration management tools, are more easily provided in the context of a single, uniform OMS. Alternatively, multiple OMSs permit the efficiencies of problem-domain-specific services and facilitate the utilization of an existing tool base. The quality of a PSE is directly related to the quality of the tools offered. The quality of the OMS may be a prerequisite for tool quality, but certainly is not a guarantor of the PSE quality. Pragmatically, it is necessary to resort, at least in part, to existing tools to populate the PSE and to ease the user migration to a PSE based on a more sophisticated OMS approach. Hence, the capability to utilize existing tools is an important consideration. Cost and availability considerations may therefore make it necessary to adjust to the existence of multiple OMSs, even though a single OMS might be more appropriate in the long run.

There is a related architectural question in positioning the OMS within the PSE: Is the OMS a "hub" service of the PSE on which all tools are layered (currently the prevalent perception), or is the OMS merely another, albeit rather special, tool that other tools can integrate with at their own choosing? Fundamental requirements on the OMS differ considerably depending on which of

these models is chosen. Generally, the arguments for a single OMS also speak in favour of a hub architecture. Similarly, opting for multiple, disjoint OMSs can obviate the question of whether there is an OMS layer in the PSE architecture.

Following are individual OMS requirements issues. Frequently, these issues relate to the single versus multiple OMS question.

2.1 Compile-time versus Run-time Type Checking

Type-checking at tool compilation to ascertain the validity of operations on OMS-administered objects has the benefits of early detection of errors and potentially minimizing run-time overhead. However, conventional languages cannot prevent malicious subversion of compile-time type checks; more restrictive languages and execution models would be required to prevent such subversion. Since integrity and secrecy of persistent data needs to be guaranteed, having both early detection of errors and integrity of persistent data will, in most environments, mean having both compile-time and run-time type checking.

Type checking of data in the object base can be performed during tool compilation, if all data types are known at compile time. Knowing all types at compile-time implies that either (1) the OMS is "closed" (no new types will be added or existing types modified after tool compilation), or (2) source code for all tools is available, allowing new types or type modifications through tool recompilation.

If the PSE is to be extensible, so that new tools and their types can be installed to share an OMS with existing tools, option (1) is undesirable. Alternatively, if multiple, distinct OMSs are acceptable, new OMSs could be added to the PSE to accommodate the introduction of new types for new tools.

Option (2) is unrealistic: Source licences for all (commercial) tools in a PSE would be prohibitively expensive. Further, type changes in a large PSE are perceived to be too common to make recompilation of all tools in that environment a viable approach.

2.2 Compile-time versus Run-time Schema Evolution

Compile-time type checking implies that tools must be recompiled if the schema is to evolve. Run-time type checks may rely on a translated form of typing information in which the schema itself requires a form of compilation before use. Both of these situations may be referred to as requiring "compile-time" schema evolution. Alternatively, if the OMS allows schema modifications without such a compilation step, it is said to permit run-time schema evolution.

Compile-time schema evolution generally implies down-time for the OMS to which the schema applies: potentially, operations on all objects must be suspended until the new schema has been installed. In an environment of multiple OMSs, each specific to one or a few functions, such an approach may be feasible (and is typically used in data-base applications). For a large, perhaps distributed IPSE based on a logically central OMS, such down-time is unacceptable. An IPSE represents an expensive investment which must have a high utilization.

Further, delaying (iterative) type evolution until down-time paralyzes the work of tool developers on the system.

Presently the prevalent, but not unanimous, perception is that the schema and type definitions need to accommodate evolution with all but unnoticeable interference with the continuous availability of the OMS.

2.3 Single versus Multiple Language Bindings

An OMS intended for a specific problem domain may require only one binding, for a language also specific to that domain. A more general purpose OMS intended to support diverse applications from diverse developers will have users requiring bindings to multiple languages.

2.4 Variety of Data Models versus Ease of Data Sharing

Multiple OMSs in the PSE permits selection of an OMS whose data model is customized to an application and whose services are optimized for that application. However, such diversity of data models and services is a substantial obstacle to the sharing of data between tools, limits the utility of active data facilities (*e.g.*, triggers) and virtually eliminates sharing and tight integration with generic activity-controlling processes (*e.g.*, with configuration or process management tools).

The choice of a data model in a single, multiple, or nested OMS is also quite controversial. Typed Entity-Relationship-Attribute (ERA) models seem to be the currently prevalent choice and farthest developed. However, object-oriented approaches, tying and restricting the availability of general operations on administered objects directly to the type of the objects carry considerable promise, even though they are as yet less established than ERA approaches. Finally, several OMSs have adopted relational data base technology as their underlying implementation.

A point of consensus is that the model's typing should support an inheritance capability so that operations and tools can be applied at appropriate abstraction levels without knowing irrelevant details of the objects operated upon.

A blended solution of nested OMSs may be desirable, in which a single underlying OMS permits substantial data sharing at a coarse granularity, but also the efficiencies of specialized object bases.

2.5 Tool Migration versus Integration

Populating a PSE with tools is an expensive undertaking, made more economically feasible if existing tools can be utilized without adapting them to the data model of a single, specific OMS. Transitioning users to the PSE is also made easier if existing tools continue to operate in the PSE. The benefits of tool integration (through data sharing and other forms of tool interaction) and tool portability (made possible through adoption of a specific kernel interface) may justify the cost of tighter integration in the long run. Nevertheless, the OMS

should make it easy to migrate existing tools without major changes into the PSE.

2.6 Persistency Models versus Performance

A commonly experienced problem with generic OMSs is their throughput performance characteristics. Lacking performance is generally attributed to the high cost of accessing and updating persistent data with the implied need for synchronization of accesses and for run-time validity checking. While there is room for performance improvement in most, if not all, current OMS implementations, it may also be conjectured that OMSs have chosen inappropriate models in dealing with persistency by postulating immediacy of the program-external availability of changes to persistent data. Today's technology is easily capable of supporting high-throughput in a memory-based, very sophisticated OMS, but is conspicuously weak in supporting a much simpler OMS on persistent data.

A primary consequence of these performance issues is that fine granularity of objects in a generic OMS may not be achievable in practice with the current persistency models.

2.7 Composite Objects

A need has long been identified to allow operations on groups of objects, as if they were a single entity for the purpose of the operation. At the same time, it has been realized that the grouping is not necessarily static, *i.e.*, the same for all operations. To-date no simple paradigm that captures both these requirements has gained significant acceptance. This lack is quite unfortunate, since, if such aggregation were applied in the contexts of access synchronization and control and of the persistency model, performance problems might be lessened if not solved.

3 Perceptions for the Future

Current research in process management support indicates that considerable opportunities exist for integrating process and object management. An OMS can play a pivotal role in implementing process management support along several dimensions: First, the process description can conceivably be represented utilizing the OMS capabilities. Thus, examination, manipulation and operation of the process description can be achieved largely by the OMS interfaces. Second, active components of the OMS model such as triggers can be used to implement process management functions. Integration of process and object management along such lines might open synergistic opportunities for new tool functionality not feasible in either isolated domain. Again, uniformity of the OMS would substantially ease the generation of such capabilities.

4 Concluding Observations

The following general observations apply to OMS theory and experiment:

- Many sophisticated requirements have been postulated for OMSs. Those discussed here may have widest consensus, but are not exhaustive.
- Many conceptual solutions meeting those requirements have been proposed.
- Experimental data allowing evaluation of the completeness and utility of the requirements and the solutions, to the extent it exists, is poorly disseminated.
- The complexity and cost of integrated PSEs impedes full-scale experiments. The solutions referred to above include third and fourth generations of systems that have yet to be effectively evaluated, even in their predecessor versions. This is particularly unfortunate, given that many OMS requirements issues center on the extent to which strong support of integration through the OMS should be pursued.

From the preceding points, it would appear that more emphasis should be placed on OMS experimentation, evaluation and dissemination of the results. Theoretical efforts in defining requirements and solutions are wanting in empirical validation. Continuing theoretical efforts should facilitate experimentation to the degree that existing proposals were unified. Given the expense of IPSE implementation and evaluation, it is likely that adequate full-scale experimentation is only possible in consortium efforts and for a very few solutions. Without such experimentation, the OMS research runs the danger of extrapolating existing models and imposing additional requirements on OMSs, before the viability of these existing models has been proven and the limitations induced by performance requirements have been sufficiently explored and reflected in the models.