FEATURE ARTICLE

Integrating Object Technology with M:
Candid Answers to Commonly Asked Questions

by Terry L. Wiechmann, John A. McManamon and Jerry E. Goodnough

ESI Technology Corporation has devoted the last 5 years to integrating Object Technology with M Technology. Within that time frame, we've tried to share what we've learned with the M community. By actually prototyping our ideas and then sharing them with the MUMPS Development Committee (MDC), a good OO standard has emerged for M. In keeping with this spirit, we would like to share some questions we've received while giving classes and presentations on Object Oriented Concepts and Object Oriented M Programming.

Questions and Answers

Q: What is Object Technology?

A: Object Technology (OT) is the name used to describe a methodology for organizing data and presenting information. First conceptualized in the early 1960's, in its purest definition it represents an "alternate universe" to relational technology. OT is being acknowledged as the next major paradigm shift in Information Technology. The term is used to encompass all aspects of Object Orientation.

Q: What is an Object?

A: An Object is the basic conceptual building block of the methodology. It is the atom of the Object universe. Fundamentally, it is comparable to an M symbol table where access to that data is only permitted through a well-defined interface - not directly.

Q: What is so special about Object Technology?

A: It is a proven way to manage the increasing cost of developing, deploying, and supporting increasingly more complex information systems. Object Technology not only allows for applications to be developed faster, it also reduces support time and problems.

Q: Isn't M already Object Oriented?

A: This is our favorite question. The answer is NO! There are three fundamental concepts that must exist for a system to be truly Object Oriented. First, the concept of encapsulation must be enforced. **Encapsulation** means that an object has a well-defined interface for exposing its data to the outside. Second, the system must display polymorphic behavior. **Polymorphism** (many forms) enables common message names sent to different objects to elicit totally different behavior. Third, objects must be able to inherit properties, methods and events from their ancestors. **Inheritance** means organizing object definitions into hierarchies and permitting properties and behavior to be shared by an object's descendants.

Q: What does Object Orientation do for M?

A: In a word, it brings **order** to application development. The programmer starts out with a well-defined, generic structure and consequently, a common foundation. The paradigm demands enforcement of its rules and features. The results are applications that are all formed on a common, well-defined foundation.

Q: How does Object Technology speed up application development?

A: The answer can be given in two words: *adaptation* and *reuse*. Initially adapting to OO takes longer, mostly due to the "unlearning" process, that is, making the transition from traditional procedural thinking to Object Oriented thinking. However, once the transition is made, you realize that the barriers you had to live with in procedural programming have now largely disappeared. The world you are trying to model and the environment you are using to model it are now synchronized - an object is an object!

Reuse is a well-advertised benefit of OO. If you start with a development environment that offers Foundation and Application Framework Classes (predefined object definitions), you will benefit immediately through reuse and consequently the development process will accelerate. Good OO development environments will offer features that augment reuse, permitting the programmer to program for differences, not recreate the wheel.

Q: How does Object Technology reduce support time and problems?

A: The phrase "Objects know what they are and what they can do" answers this question indirectly. Typical, traditional M applications consist of data stored in global arrays with a mountain of code that acts upon this data. Casually examining the code of one of these systems will reveal a commonly accepted organization of IF - ELSE statements that often are asking "Who am I and what am I doing now?" We all know that this code often springs logical leaks and is the source of numerous support problems.

Objects on the other hand, contain both the data and the code that acts on that data. The object knows what it is and what it can do! As a consequence, a great deal of superfluous code disappears along with the incumbent support problems.

Another phrase, "If you want something from me, go to the front door and ask!", offers another reason for reduced support. Encapsulation means that an object's data can only be accessed through a predefined interface (the front door). Indiscriminately accessing data across object boundaries is forbidden. Code (in the form of methods, properties, and events) stands guard at the door. Each method or event acts upon the data to display some discretely defined behavior. Each property exposes a piece of data to the caller. Because this code is localized and discrete, bugs are confined to a very small space. The famous "ripple effect" does not get very far - it is confined to the object's boundaries.

Q: I can do that now in M! Why should I go to Objects?

A: It's important to realize that OO does not "computationally" add any more power than traditional programming. Given a particular application developed in OO, one can develop a similar application using current techniques. The question that needs to be asked is "What about version two, three, and four?" What happens when the system becomes extremely complex? Studies have shown that most traditional approaches tend to fall apart once complexity hits a certain level. OO is proving to push that barrier much farther out. OO flattens out the maintenance versus complexity curve.

Q: Isn't OO just another way of saying Structured Programming?

A: An important point to remember about OO is this "new" paradigm has retained the proven techniques of the past. In addition to encouraging structured programming techniques, the infrastructure of a true OO system incorporates structure. Additionally, the concept of hiding data (encapsulation) is enforced. However, OO does not stop there, it adds the other necessary concepts of polymorphic behavior and inheritance. A well designed OO environment offers a structure where a real world model of an application can be designed in an intuitive fashion.

Q: M is simple to use and I can prototype a system very fast. Will I be giving that up?

A: No! In fact OO augments those capabilities. Good OO development environments contain libraries of reusable objects, that is, Classes that define objects. If you are going to build a house, you shouldn't have to make the bricks and cut the lumber before starting. Reusable objects accelerate prototyping. Additionally, what could be simpler than sending a message to an object and having it perform an operation.

On the other hand, if you are building the object definitions yourself, then in all honesty, it usually takes more time to prototype a solution. OO requires that more time be spent in the Analysis and Design phase of development than traditional approaches. Fast prototyping has all too often been used as a justification for ignoring Analysis and Design.

Q: What makes M a good candidate for Object Orientation?

A: M is a natural platform for Object Technology because it shares some of the core features of an object system. M is hierarchical and so is an object environment. M is fundamentally interpretive, pushing decision making down to runtime, an essential feature of objects. M has supported persistence from the beginning, a required feature of an object oriented database. So, in a very complementary fashion, M technology has developed capabilities for persistent objects, for dynamic binding, and for runtime execution, which are all crucial requirements for a functioning Object environment. What it does not support is the primitive object type, encapsulation, polymorphic behavior and inheritance. The emerging OOM standard lays a foundation for the addition of these features.

The questions and answers touched on above are typical. We hope we have answered them to your satisfaction.

Terry L. Wiechmann is President of ESI Technology Corporation, John A. McManamon and Jerry E. Goodnough are the principal architects and implementors of EsiObjects, ESI's Object Oriented M product.