Preface

Objects, object-oriented techniques, and object technology have altered the software development landscape for software professionals and, more slowly, altered the cost, quality, and timeliness of modified and new systems for users and customers. Objects are the latest and most effective means so far for creating software that is easy to develop, relatively error-free, portable, reusable, and serves as a continually evolving platform for yet more advanced software capabilities.

This issue contains an introductory essay on the evolution of object technology, six papers on various facets of object technology and its use, and a technical note on supertypes and subjects. We are indebted to J. R. Babb of the IBM Software Solutions Division in Somers, New York, for his contributions to the early planning and coordination of this issue and to G. Radin of the IBM Research Division in Yorktown Heights, New York, for his noteworthy efforts in coordinating and developing this issue.

The existence of object technology owes much to a long line of preceding technologies that evolved into what we see and use today. Object technology represents the latest and best approach to achieving software productivity, quality, maintainability, reuse, and effective development. Radin, in an introductory essay, describes aspects of object technology that demonstrate its power for software development and developers, while reflecting on the historical precedents and evolutionary trail.

The ability to create information systems that serve enterprises depends on both an inside view of those systems, such as is supplied by architecture and engineering, and an outside or user view. As McDavid shows in his paper, this outside view is fundamentally dependent on an understanding of the enterprise as an information system, communicating through domain-specific languages. The result is business language analysis. The effect is improvement in the link between the application domain (the enterprise) and domain-sensitive object technologies.

Budinsky et al. provide the architecture for and a description of a software tool for automatic translation of object-oriented design patterns into code, based on design patterns taken from the current literature on the subject. The architecture emphasizes speed of tool development and enhancement, flexibility in adding and changing functions, ease of specification of the design-to-code translations, and ease of use of the tool and its results, among other goals. The tool is new and much remains to be learned from users and incorporated from the ever-expanding breadth and depth of design patterns.

The storage of application data in one or more of the many database management systems has become problematic for the application programmer and is the subject of a paper by Reinwald et al. The authors present a shared memory-resident cache as an extension to IBM's DB2* Common Server for AIX* that provides a means for storing objects within a relational database management system. While a number of types of data are discussed, the paper uses C++ objects as an example of how common data storage can be achieved without affecting the underlying objects.

Benantar, Blakley, and Nadalin explore the requirements for and architecture of the run-time Object Security Service (OSS), a part of IBM's Distributed SOM (System Object Model) environment. OSS handles authentication, authorization, client/server association, and other security functions, while taking advantage of existing security mechanisms. OSS provides these services to end users and application developers transparently, seamlessly, and independently of other underlying and coexisting services.

One difficulty in achieving high-quality software is the lack of software metrics that can be viewed in

real time, as the software is being developed. Burbeck approaches this problem in the broad context of general real-time metrics and then in the specific context of complexity metrics for object-oriented software development in Smalltalk. The author searches out metrics that can be readily determined from code and are measurable in real time, resulting in seven complexity metrics. Advice is given regarding the value and use of these metrics in practice.

Davis, Grimes, and Knoles address the technological response to the need for international use of text in global applications that have only one physical, binary form. They further consider the localization aspects (language, time, date formats, etc.) that must be adequately handled by these global applications. Their context for discussion is the globalization features of Taligent's CommonPoint** application system for development of documents and object-oriented applications, and the Unicode** international character encoding standard.

In a technical note, Harrison et al. describe their recent work on combining the capabilities of dynamic supertypes and subjects (from subject-oriented programming) that they have separately developed. This first look at the combination shows how specification and development of object-oriented software would be enhanced, especially for large, complex systems. The combined approach is intended to provide a natural way to encompass the entire software development life cycle.

The next issue of the ***Journal*** will be a special double issue (Volume 35, Numbers 3 and 4 as one issue) on the current projects of the Massachusetts Institute of Technology (MIT) Media Lab, famous for its work on multimedia and now exploring many new fields of future-oriented computerization.

Gene F. Hoffnagle
Editor

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Taligent, Inc. or Unicode, Inc.