Frameworks are specialized for a narrow range of applications; each model of interaction is domain-specific, e.g., designed to solve a narrow set of problems. Frameworks evolve over long periods of time. A framework is more than a class hierarchy. It is a miniature application complete with a dynamic as well as a static structure. It is a generic application and can be reused as the basis for many other applications. This topic has captured a great deal of attention in recent years, since frameworks leverage, in practical terms, capital-intensive software investment through reuse, and higher-level application programming interfaces, so that applications can be developed many times faster.

Despite dramatic increases in computing power, the design and implementation of complex software remains hard. Moreover, the growing heterogeneity of hardware/software architectures and diversity of operating system and communication platforms make it difficult to reuse existing algorithms, detailed designs, interfaces, or implementations directly. The emerging focus on object-oriented enterprise and application frameworks (OOAFs) in the OO community offers software developers both a new vehicle for reuse and a way of capturing the essence of successful architectures, components, policies, services, and programming mechanisms. By providing reusable skeletons on which to build new applications, frameworks can save countless hours and thousands (even millions) of dollars in development costs. Currently, OOAFs are a very important issue for the software industry and academia because software systems are becoming increasingly complex. I believe that OOAFs will be at the core of the technology of the twenty-first century.

Additional Key Words and Phrases: agent-oriented frameworks, aspect-oriented frameworks, built-in test reuse, class libraries, components, constraints satisfaction, design patterns, domain analysis, domain-specific frameworks, enterprise frameworks, framelets, framework architecture, framework documentation, frameworks and pattern languages, role object pattern

INTRODUCTION AND DEFINITIONS

Object-Oriented Application Frameworks (OOAFs) are quickly becoming strategic assets for organizations across all business sectors, perhaps due to the recognition that re-engineering failures may be occurring at a rate as high as 70% [Fingar 1996]. Thus, despite dramatic improvements in computing power and software development tools,
the design, implementation and maintenance of complex software remains difficult, expensive, and risky.

A framework is a reusable, "semicomplete" application that can be specialized to produce custom applications [Johnson and Foote 1998]. In contrast to earlier OO reuse techniques based on class libraries, frameworks are targeted for particular business units (such as data processing or cellular communications) and application domains (such as user interfaces or real-time avionics) [Fayad and Schmidt 1997].

Frameworks take advantage of all three of the distinguishing characteristics of object-oriented programming languages: data abstraction (like an abstract data type, an abstract class represents an interface behind which implementations can change); polymorphism (the ability for a single variable or procedure parameter to take on values of several types); and inheritance (inheritance makes it easy to make a new component) [Fayad et al. 1999].

A framework describes the architecture of an object-oriented system. It is represented by a set of classes (usually abstract), one for each kind of object, but the interaction patterns between objects are just as much a part of the framework as the classes [Fayad et al. 1999].

Successful frameworks evolve and spawn other frameworks. One of the first user interface frameworks influenced by Model/View/Controller was MacApp, which was designed specifically for implementing Macintosh applications [Schmucker 1986]. Frameworks are not limited to user interfaces but can be applied to any area of software design. They have been applied to VLSI routing algorithms [Gossain 1990]; to hypermedia systems [Meyrowitz 1986]; to structured drawing editors [Vlissides and Linton 1989; Beck and Johnson 1994]; operating systems [Russo 1990]; psychophysiological experiments [Poonte 1998]; network protocol software [Hueni et al. 1995]; and manufacturing control [Schmidt 1997], to mention a few.

Frameworks do not even require an object-oriented programming language. For example, the Genesis database system compiler is a framework for database management systems [Batory et al. 1999] as well as a tool for specifying how DBMSs are built from the framework. Genesis does not use an object-oriented language, but rather a macroprocessor and conditional compilation to implement an object-oriented design in C.

**CLASSIFYING APPLICATION FRAMEWORKS**

We categorize application frameworks into three categories:

(1) **System infrastructure frameworks** simplify the development of portable and efficient system infrastructure such as GUIs, Choice operating system [Campbell and Islam 1993] and communication frameworks [Schmidt 1997], and frameworks for user interfaces and language processing tools. System infrastructure frameworks are primarily used internally within a software organization and are not sold to customers directly.

(2) **Middleware integration frameworks** are commonly used to integrate distributed applications and components. They are designed to enhance the ability of software developers to modularize, reuse, and extend their software infrastructure to work seamlessly in a distributed environment. Common examples include ORB frameworks, message-oriented middleware, and transactional databases.

(3) **Object-oriented enterprise frameworks** address broad application domains (such as telecommunications, avionics, manufacturing, and financial engineering [Birrer 1993]), and are
the cornerstone of enterprise business activities [Hamu and Fayad 1998]. Relative to system infrastructure and middleware integration frameworks, enterprise frameworks are expensive to develop and/or purchase. However, they can provide a substantial return on investment because they support the development of end-user applications and products directly. In contrast, system infrastructure and middleware integration frameworks focus largely on internal software development. Although these frameworks are essential to rapidly create high-quality software, they typically do not generate substantial revenue for large enterprises. As a result, it is often more cost effective to buy system infrastructure and middleware integration frameworks rather than build them in-house [Hamu and Fayad 1998].

Regardless of their scope, frameworks can also be classified by the techniques used to extend them, which range along a continuum from whitebox frameworks to blackbox frameworks. Whitebox frameworks rely heavily on OO language features like inheritance and dynamic binding in order to achieve extensibility [Fayad et al. 1999]. In contrast, blackbox frameworks are structured using object composition and delegation rather than inheritance.

STRENGTHS AND WEAKNESSES OF APPLICATION FRAMEWORKS

When used in conjunction with patterns, class libraries, and components, OO application frameworks can significantly increase software quality and reduce development effort. However, a number of challenges must be addressed in order to employ frameworks effectively. Companies attempting to build or use large-scale reusable frameworks often fail unless they recognize and resolve challenges such as development effort, learning curve, integratability, maintainability, validation and defect removal, efficiency, and lack of standards [Fayad and Schmidt 1997].

APPLICATION FRAMEWORKS VERSUS OTHER REUSE APPROACHES

Frameworks are closely related to other approaches to reuse, including

(1) Design patterns: A pattern represents a recurring solution to a software development problem within a particular context. Patterns and frameworks both facilitate reuse by capturing successful software development strategies. The primary difference is that frameworks focus on reuse of concrete designs, algorithms, and implementations in a particular programming language. In contrast, patterns focus on reuse of abstract designs and software architectures. Frameworks can be viewed as a reification of families of design patterns. Likewise, design patterns can be viewed as the micro-architectural elements of frameworks that document and motivate the semantics of frameworks in an effective way. When patterns are used to structure and document frameworks, nearly every class in the framework plays a well-defined role and collaborates effectively with other classes in the framework.

(2) Class libraries: Frameworks can be viewed as extensions to object-oriented class libraries. However, they are distinguished from class libraries in the following ways: Frameworks define “semicomplete” applications that embody domain-specific object structures and functionality. Components in a framework work together to provide a generic architectural skeleton for a family of related applications. Complete applications can be composed by inheriting from and/or instantiating framework components. In contrast, class libraries provide a smaller granularity of reuse. More-
over, class library components like classes for strings, complex numbers, arrays, and bitsets are typically lower-level and more domain-independent. Frameworks are active and exhibit “inversion of control” at runtime. Class libraries are typically passive, i.e., they perform their processing by borrowing threads of control from self-directed application objects. In contrast, frameworks are active, i.e., they control the flow of control within an application via event dispatching patterns such as reactor and observer. The “inversion of control” in the runtime architecture of a framework is often referred to as the Hollywood principle, i.e., “don’t call us, we’ll call you.”

(3) Components: Frameworks primarily address software development concerns. Components primarily address end-user concerns. Frameworks can be used to develop components. Likewise, components can be used in blackbox frameworks.

A BRIEF SYNOPSIS OF THE ELECTRONIC SYMPOSIUM

The articles in this symposium describe how object-oriented application frameworks provide a powerful vehicle for reuse and extensibility, as well as a way to capture the essence of successful documentation, architectures, patterns, components, policies, services, and programming mechanisms. In addition, the symposium’s articles cover the entire lifecycle of developing and adapting application frameworks and provide lessons learned from practical application framework developments and adaptations. The articles are categorized into six different groups:

(1) Framework perspectives: This category has three articles.

In “Frameworks and Pattern Languages: an Intriguing Relationship,” D. Brugali and G. Menga compare application frameworks and pattern languages and show that frameworks and pattern languages complement each other in the software development process.

In “Object-Oriented Framework-Based Software Development: Problems and Experiences,” J. Bosch et al. describe and organize a number of framework problems into four categories, i.e., framework development, usage, composition, and maintenance. The most relevant problems and experiences are presented for each category.


(2) Framework development and adaptation: This category has nine articles.

It leads off with “Domain-Driven Framework Layering in Large Systems” by D. Bäumer et al. These authors draw on their experience developing large-scale banking projects to present concepts and techniques for domain partitioning and framework layering in order to overcome difficulties in framework construction.

“Framelets—Small and Loosely Coupled Frameworks” by W. Pree and K. Koskimies proposes framelets as small architectural building blocks that can be easily understood, modified, and combined. It presents a sample framelet implemented in Java.

“On Built-In Test Reuse in Object-Oriented Framework Design” by Y. Wang et al. extends software reusability from code and architecture to built-in tests (BITs) in object-oriented framework development and addresses methods for embedding BITs at object and object-oriented framework levels.

objects as language constructs give precise means for controlling the framework extensions in statically checkable ways.

“Deferring Design Decisions in an Application Framework” by J. Carey and B. Carlson explains that the IBM San Francisco application framework addresses this issue by implementing specific design patterns that exhibit varying degrees of design deferral.

“Pattern Density and Role Modeling of an Object Transport Service” by D. Riehle et al. describes patterns as role models and indicates that a pattern-based framework description using role models makes the design more comprehensible than a description that solely focuses on classes.

“Developing Object-Oriented Frameworks Using Domain Models” by M. Aksit et al. presents an integrated approach to model the domain knowledge related to a framework and to map the identified domain models into object-oriented concepts, and indicates that deriving a framework from the related domain knowledge considerably reduces the amount of framework refinement time.

“Framework Extraction with Domain Analysis” by G. Succi et al. describes the main features of the Sherlock domain analysis method and presents a case study on the development of a graphic user interface framework for business and management information systems.

“The Framework Approach for Constraint Satisfaction” by P. Roy et al. outlines the design of such a framework and stresses the various advantages of the constraint satisfaction programming (CSP) approach compared to others. The authors classify the various approaches to this problem into three categories: the language approach, in which only basic mechanisms are proposed to the user; the library approach, in which the user may pick up pre-designed algorithms off-the-shelf; and the object-oriented framework approach, an intermediary position between languages and libraries.

(3) Framework documentation: This category has three articles.

“Documenting Framework Behavior” by N. Soundarajan points out the need for a new approach to specify the behavior of frameworks, outlines a possible new approach, and briefly considers how an application developer can combine a framework specification of the type proposed with appropriate information about the code to arrive at a specification of his particular application.

“A Framework for Framework Documentation” by G. Butler et al. presents a task-oriented framework for framework documentation, and compares some major framework documentation approaches, in light of the framework above, identifying possible deficiencies and highlighting directions for further research. The authors point out that their framework is based on first identifying a set of framework (re)use cases, which are decomposed into a set of elementary engineering tasks. Each such task requires a set of documentation primitives, enabling us to specify a minimal set of documentation primitives for each framework scenario.

The “Framework Description Using Concern-Specific Design Pattern Composition” by A. Silva et al. proposes an approach to object-oriented framework description based on using design patterns for describing concern variations and on using design patterns for describing various compositions.

(4) Framework measurements: This category has two articles.

“Choosing an Object-Oriented Domain Framework” by G. Froehlich et al. outlines a three-step process to determine whether or not a framework is appropriate, and looks at the limitations, the hooks, and the amount of uncertainty in using a framework.

“Evaluating Framework Architectures’ Structural Stability” by J. Bansiya evaluates the structural characteristics of framework architectures using a suite of object-oriented design metrics.
on Microsoft Foundation Classes (MFC) and Borland Object Windows (OWL) application framework systems. Results show that the most significant causes of structural change between releases are reworks in collaborations between classes, assignment of responsibilities to classes, and the addition of new classes to enhance the features/capabilities of the framework systems.

(5) **Agent-oriented technology**: This category has three articles.

“An Overview of Software Agent-Oriented Frameworks” by Z. Maamar and B. Moulin presents the structural and functional characteristics of software agent-oriented frameworks and describes their use in the context of the SIGAL project, which aims at designing an interoperable environment for georeferenced digital libraries.

“An Application Framework for Intelligent and Mobile Agents” by E. A. Kendall et al. summarizes research in designing and developing an application framework for intelligent and mobile agents. The authors points out that the framework has been employed in preliminary applications in network management and enterprise integration.

“Towards Agent-Oriented Application Frameworks” by D. Brugali and K. Sycara defines the concept of an agent-oriented application framework as a re-use technique and compares it with object-oriented application frameworks and component libraries.

(6) **Domain-specific application frameworks**: This category has twenty articles.

“An Open Framework for Reliable Distributed Computing” by B. Garbinato and R. Guerraoui gives an overview of an open object-oriented framework for reliable distributed computing, called BAST. BAST provides abstractions for building reliable distributed protocols such as atomic commitment and total order broadcast.

“An Open System Framework for Component-Based CNC Machines” by J. Michaloski et al. describes a framework for open, component-based, and manufacturing controllers. This framework can be used to build applications that range from a single-axis device to a multiarm robot.

“Towards an Open Multimedia Service Framework” by L. Fuentes and J. M. Troya describes a component-oriented framework (MultiTEL) for the development of multimedia services (MS) on top of the web. The major contribution of this paper is the application of componentware technologies to a large-scale system.

“The San Francisco Project: Business Process Components and Infrastructure” by V. Johnson and B. Rubin provides an overview of IBM’s San Francisco project as a way to simplify the move to distributed object-oriented applications. The San Francisco project includes an infrastructure for distributed object applications, business objects that are common across multiple domains, and domain-specific business process components that contain a subset of the objects and business logic needed to build a commercial application.

“A Framework Architecture for Supervision and Control Systems” by R. Capobianchi et al. presents a framework architecture supporting the development of supervision and control systems on top of CORBA. The paper also describes a collection of such frameworks as suitable CORBA domains.

“A Framework for Workflow Management Systems Based on Objects, Rules, and Roles” by G. Kappel et al. describes an object-oriented application framework for constructing workflow management systems that balance between reusability and adaptability. The underlying techniques are an object-oriented workflow model, object evolution via an integrated role model, and the support of business policies via an integrated rule model.

“Application of Frameworks in Groupware—The Iris Group Editor Environment” by M. Koch and J. Koch discusses
groupware as one application domain where tailorability is a critical issue, and present the framework-based architecture of our group editor application, Iris.

“A Framework for Isolating Connection Exception Management” by P. Pal presents an abstract, object-oriented application framework for isolating connection-related failure management from the main application.

“Navigating Between Objects: Lessons from an Object-Oriented Framework Perspective” by G. Rossi et al. presents a general architecture for building computational hypermedia applications, i.e., those applications that combine the hypermedia navigational style with other kinds of computations in an object-oriented system. The authors discuss the most important design decisions behind the framework, presenting them as a set of micro-architectural constructs that yield a general architecture for integrating object-oriented and hypermedia applications.

“An Object-Oriented Framework for Data Parallelism” by J.-M. Jézéquel describes a framework where the data parallel codes can be encapsulated in object-oriented software components that can be reused, combined, and customized with confidence by library designers to offer application programmers easy-to-use programming models.

“PAcceptor and SConnector Frameworks: Combining Concurrency and Communication” by R. Kannan presents a single interface to these disparate, but essential, services, and thus promotes the use of these abstractions in combination with ease. The PAcceptor and SConnector frameworks encapsulate both threads (primitive for concurrency) and sockets (primitive for interprocess communication). The PAcceptors accept incoming connect requests using sockets and provide service in parallel using threads, while the SConnectors permit one or more threads to share a single connection to send data over the connection.

“HotDoc—A Framework for Compound Documents” by J. Buchner presents HotDoc, a framework for the development of editors for compound documents. HotDoc allows the construction of flexible documents consisting of dynamic parts and introduces a new type of document. A document is not only a static sequence of texts, but an interface to small applications (parts). The author points out that programmers can easily implement new parts by using abstract classes of the framework. HotDoc is implemented in VisualWorks Smalltalk.

“The Zyper Open Hypermedia Framework” by S. Demeyer et al. discusses the use of open hypermedia to provide a seamless integration of framework documentation, design, and implementation. In particular, it shows how computational hypermedia links can ensure the consistency between the source code and a framework cookbook.

“Experiences and Issues with SEMATECH’s CIM Framework” by P. Whelan argues that large frameworks, such as the CIM, must be able to support a complex software supply chain rather than a simple supplier-user model. Issues of component granularity, extendibility, and substitutability are presented. A brief discussion on a set of proposed conformance rules is included at the end of the paper.

“Experiences with an Object-Oriented Framework for Distributed Control Applications” by R. Harinath et al. describes the authors’ experiences with the design and implementation of an object-oriented framework for distributed control applications. The paper describes the salient features of this framework, explores the challenges faced in integrating commercial off-the-shelf (COTS) object-oriented products into the framework, and discusses how this affected the design and implementation.

“Adaptive Object-Oriented Filtering Framework for Event Management Applications” by E. Al-Shaer et al. describes a flexible event-filtering framework that can be efficiently adapted to different domain-specific requirements with minimal development
effort and presents examples of using the event-filtering framework for developing event-management applications in different domains. Event filtering is an essential element in event-management applications.

"Using Object-Oriented Concepts for 3D Visualization and Validation of Industrial Scenarios" by V. Luckas and R. Dörner describes an integrated application framework into the automatic generation of 3D visualizations and animation based upon sensor, simulator, or layout-planning data. The object-oriented design of animation elements integrating geometry and individual object-specific behavior is the technical innovation of this paper. Animation elements build an object-oriented framework as the elements may be customized for a specific application.

"Developing Flexible and High-Performance Web Servers with Frameworks and Patterns" by D. Schmidt and J. Hu illustrates how frameworks and patterns address complexities that arise in the design and implementation of high-performance distributed software systems. These complexities are both inherent (e.g., latency reduction and throughput preservation) and accidental (e.g., the continuous reinvention of key concepts and components). This paper explains how complexities in the development of high-performance web servers can be alleviated with the use of design patterns and object-oriented application frameworks.

"The Five-Module Framework for Internet Application Development" by E. Ebner et al. presents the five-module architecture of a framework for Internet application development, the DIWB (Distributed Interactive Web-Site Builder), and shows how this architecture has been implemented, exercised, and deployed on the web. The five-module architecture separates the application into five modules: presentation, UI components, business logic, data management, and system infrastructure.

"Designing an Aspect-Oriented Framework (AOF) in an Object-Oriented Environment" by C. Constantinides et al. discusses separation of concerns as the heart of software development. The framework approach is promising, as it seems to be able to address a large number of aspects (and applications) as long as the relationships of components and aspects (as well as the aspect interrelationships) are clearly defined. A clear definition of these inter-relationships is achieved through the use of preconditions and postconditions. In AOF, the introduction of a new specification (aspect) must be accompanied by a set of rules that will ensure the integrity of the semantics of the system. These rules are expressed as preconditions, postconditions, and the order of activation of aspects.

CONCLUSIONS

The articles in this symposium illustrate that object-oriented enterprise and application frameworks will be at the core of leading-edge software engineering technology in the twenty-first century. The shifting focus on software-component, enterprise, and application frameworks in the object-oriented community offers software industries a powerful vehicle for reuse and extensibility, a better way for engineering software, and a means to capture the essence of successful patterns, architectures, policies, components, and programming mechanisms.

The good news is that enterprise and application frameworks are becoming mainstream, and that developers at all levels are increasingly adopting (and succeeding with) application framework technologies.

REFERENCES


European Conference on Object-Oriented Programming (Bologna, Italy, July), 139–149.