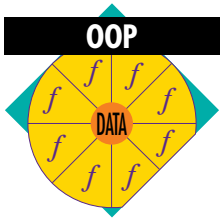
Building Object-Oriented Frameworks

Part 1



By Deborah Adair

By using frameworks1, Taligent®, Inc.—an independent joint venture of Apple® Computer, IBM, and Hewlett-Packard—is realizing the full promise of object technology. A framework defines the behavior of a collection of objects, providing an innovative way to reuse software designs and code. Part 1 of this two-part series describes different types of frameworks and how they are used and addresses organizational concerns that impact framework development. Part 2 will outline a process for identifying and designing frameworks.

This article presents a process for developing frameworks and highlights four important guidelines:

- Derive frameworks from existing problems and solutions
- Develop small, focused frameworks
- Build frameworks using an iterative process driven by client[2] participation and prototyping
- Treat frameworks as products by providing documentation and support, and by planning for distribution and maintenance

While the process and techniques discussed are ideal for developing in the Taligent CommonPoint™ application system, they can also be applied to other object-oriented programming projects.

This article is intended primarily for software developers and designers in commercial, corporate, and higher education software development organizations. However, hardware designers, strategic technology planners, and technical managers might also find it useful.

Understanding Frameworks

A framework captures the programming expertise necessary to solve a particular class of problems. Programmers purchase or reuse frameworks to obtain such problem-solving expertise without having to develop it independently.

Framework Development

Developing a framework differs from developing a stand-alone application. A successful framework solves problems that appear different from the problem that justified its creation. The problem-solving expertise must be captured so that it is independent of both the original problem and the future solutions in which it is used; however,

Deborah Adair

[1]A framework is a set of prefabricated software building blocks that programmers can use, extend, or customize for specific computing solutions. Taligent has designed frameworks for system software functions, such as networking, multimedia, and database access. With frameworks, software developers do not have to start from scratch each time they write an application. Frameworks are built from a collection of objects, so both the design and code of a framework can be reused.

[2]This article is written from the viewpoint of framework developers. We use the term "client" to refer to developers who might use a framework that you developed.

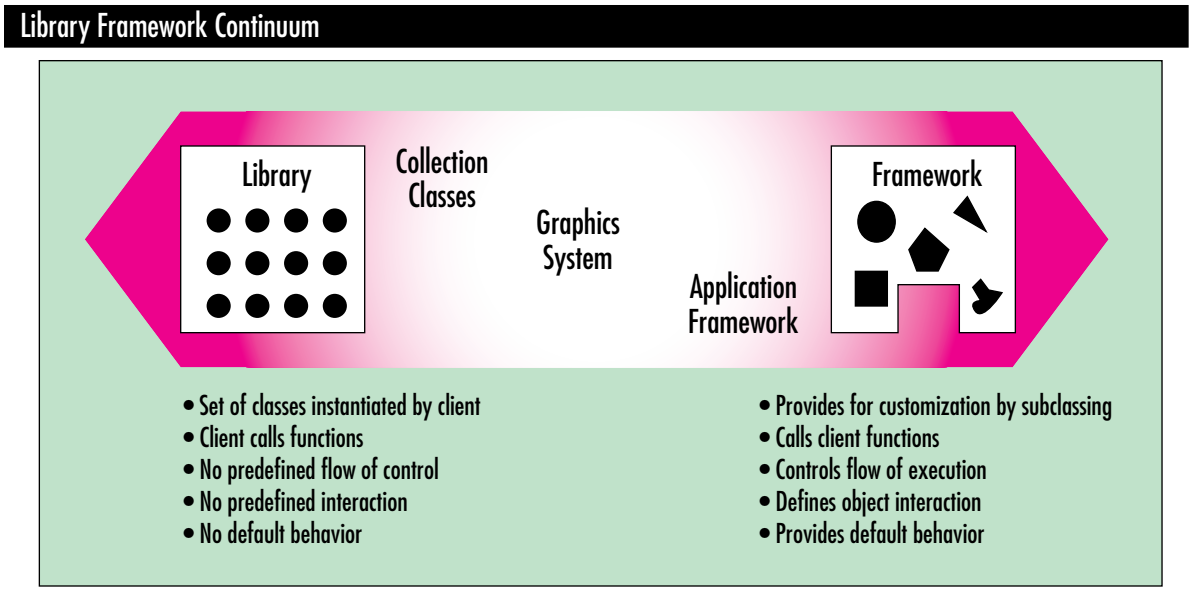


Figure 1. Library framework continuum

each program that uses the framework should appear to be the one for which it was designed.

Developers must clearly identify the class of problem that a framework addresses. For your clients to adapt the framework to new problems, they must understand the solution the framework provides and how to incorporate it into their programs. Because others have to understand how to use your frameworks, it is critical that you follow good software design practices.

Describing Frameworks

While clear differences exist between class libraries and frameworks, some libraries exhibit framework-like behavior, and some frameworks can be used like class libraries. As shown in Figure 1, this can be viewed as a continuum, with traditional class libraries at one end and sophisticated frameworks at the other.

Frameworks can be characterized by the problem domains that they address, as well as by their internal structure.

Framework Domains

The problem domain that a framework addresses can encompass application, domain, or support functions.

- **Application frameworks** encapsulate expertise applicable to many programs. These frameworks encompass a horizontal slice of functionality that can be applied across client domains. Current commercial Graphical User Interface (GUI) application frameworks, which support the standard functions required by all GUI applications, are one type of application framework. (The Apple MacApp™ system and Borland's OWL™ system are two such frameworks.)
- **Domain frameworks** encapsulate expertise in a particular problem domain. These frameworks encompass a vertical slice of functionality for a specific client domain. Examples of domain frameworks include a control systems framework for developing manufacturing control applications, a securities trading framework, a multimedia framework, or data access framework.
- **Support frameworks** provide system-level services, such as file access, distributed computing support, and device drivers. Application developers typically use support frameworks directly or after modification by systems providers. Support frameworks can be customized, such as developing a new file system or device driver.

The Taligent CommonPoint application frameworks extend across all three categories. Similarly, you can create frameworks that capture your own problem-domain expertise. Whether you are developing custom applications in a corporate setting or developing a suite of commercial applications as a software vendor, building and using frameworks can increase productivity. The frameworks you build will usually be domain or application frameworks.

Using the Framework

- Set box color
- Give box contents
- Draw box and contents


Drawing a default check box

```
TBox box;
box.SetColor(TRGBColor(0,1,0);
TCheckContents check;
box.SetContents(check);
box.Draw();
```

Extending the Framework

- Derive from contents class and override DrawContents


Extending the framework to draw an X instead of the default check

```
class TXMarkContent : public TBoxContent{
public:
   virtual void DrawContents(TGrafPort&port)
};
void
TxMarkContent::DrawContents(TGrafPort&port)
 {
   TLine line1(TGPoint(0,0),TGPoint(10,10))
   Tline line2(TGPoint(10,0),TGPoint (0,10))
   line1.Draw(port)
   line2.Draw(port);
}
```

Figure 2. Using and extending a framework

Framework Structures

Identifying the high-level structure of a framework makes it easier to describe the behavior of the framework and provides a starting point for designing framework interactions. For example, some frameworks are manager-driven—a single controlling function triggers most framework actions. You call the controlling function to start the framework, and the framework creates the necessary objects and calls the appropriate functions to perform a specific task.

An application framework generally uses a manager object to take input events from the user and distribute them to the other objects in the framework.

Another categorization, based on how a framework is used, is whether you derive new classes or instantiate and combine existing classes.

Architecture-driven frameworks rely on inheritance for customization. Clients customize the behavior of the framework by deriving new classes from the framework and overriding member functions.

Data-driven frameworks rely primarily on object composition for customization. Clients customize the behavior of the framework by using different combinations of objects. The objects that clients pass into the framework affect what the framework does, but the framework defines how the objects can be combined.

Frameworks that are heavily *architecture-driven* can be difficult to use because they require a substantial amount of code to be written. Purely data-driven frameworks are generally easy to use, but they can be limiting.

One approach for building frameworks that are both easy to use and extensible is to provide an architecture-driven base with a data-driven layer. Most frameworks provide ways for clients to use the built-in functionality and also modify that functionality. Typically, clients use a framework's built-in functionality by instantiating classes and calling their member functions. Clients extend and modify a framework's functionality by deriving new classes and overriding member functions.

Figure 2 shows a simple example of using and extending a framework. Suppose you have a simple framework that supports, among other things, drawing shapes. Clients might use it to draw a check box or extend it to draw other types of boxes.

Maximizing Framework Benefits

The key to maximizing the benefits of writing frameworks is to get as many developers as possible using them. Frameworks are a long-term investment; the benefits gained from developing frameworks are not necessarily immediate because framework designers need more time to create a framework than a procedural library, and clients need more time to learn a framework than a procedural library.

Managing Dependencies

Projects can often be divided into several separate frameworks and assigned to small teams. If it takes more than three or four programmers to produce a framework, it can probably be split into a set of smaller frameworks. Teams of two to four are usually more effective than teams of one, unless the single programmer is both an experienced framework developer and a domain expert.

Working with several small teams has its challenges:

- Programmers are focused on one aspect of a large project and might not understand all the interrelationships and client implications.
- Architectural consistency must be maintained across teams.
- Dependencies between frameworks can create bottlenecks.

There are several ways to alleviate these problems:

- Appoint a project architect who maintains the "big picture" and ensures that the frameworks ultimately work together.
- Follow standard design and coding guidelines.
- Decouple the frameworks by isolating the dependencies in intermediary classes.

Often when one framework requires the services of another, the connection can be implemented through an interface or server object. Then, only one object is dependent on the other framework. Until the other framework can support the necessary operations, the intermediary class provides stub code that allows the rest of the framework to be tested. Loosely coupled frameworks are generally more flexible from the client's perspective.

Designing Successful Frameworks

Successful frameworks have the following characteristics:

- **Complete**—Frameworks support features needed by clients and provide default implementations and built-in functionality where possible. Provide concrete derivations for the abstract classes in your frameworks and default member function implementations to make it easier for your clients to understand the framework and allow them to focus on the areas that they need to customize.
- **Flexible**—Abstractions can be used in different contexts.
- **Extensible**—Clients can easily add and modify functionality. Provide hooks so your clients can customize the behavior of the framework by deriving new classes.
- **Understandable**—Client interactions with the frameworks are clear, and the frameworks are well documented. Follow standard design and coding guidelines and provide sample applications that show the use of each framework. If the developers who build frameworks and those who use them follow the same guidelines, it facilitates reuse in both directions.

The most important consideration when designing frameworks is ease of use for the client programmer. From the client's perspective, an easy-to-use framework performs useful functions with no effort. The framework works with little or no client code, even if the default implementations are simply placeholders, and it supports small, incremental steps to get from the default behavior to sophisticated solutions. However, it is harder for clients to work around bad abstractions than invent their own. If you do not know how to solve a problem in a reusable way inside your framework, leave it out.

A framework does not necessarily have to meet all of these requirements to be successful; but if it does, it will be easier to convince other programmers to use the framework. When you design a framework, also look for ways to minimize the potential for client errors and enhance portability:

- Simplify clients' interactions with the framework to help prevent client errors. Make any client requirements as clear as possible in both your interfaces and documentation.
- Isolate platform-dependent code for easier portability of your framework. Designing for portability reduces the impact of porting on your clients.

This approach can also be used to isolate platform-dependent code or code that accesses a specific application's framework. To use different platforms or application frameworks, only one piece of the framework needs to be changed. Intermediaries can also be used to access legacy data or non-framework services.

Delivering Your Framework as a Product

Even if your framework will only be used internally, it must be treated like a product. Documenting your framework is an important step in the development cycle, but you must also plan how the finished product will be distributed and supported.

To use your frameworks, other developers need to know that they exist and how to access them. Unless a process is established for providing and distributing frameworks, it is difficult for other developers to use them. Reuse does not just happen; your organization must actively support and encourage it. One way to do this is to reward programmers for writing and distributing frameworks that others can use. Even more important, reward the programmers who use them.

Ideally, all frameworks are kept in a central repository, and a repository manager is responsible for notifying clients about new frameworks and updates to old ones. With a large enough repository, selecting the appropriate frameworks becomes an integral part of developing new program solutions.

To realize the benefits they can provide, your organization must commit resources to support frameworks. You must be able to assist your clients and respond to their problems and requests.

Over the lifetime of a framework, the cost of supporting it actually becomes a benefit—the support cost of one framework with three dependent applications will be less than the cost of supporting three independent applications with duplicate code. The more applications that use a framework, the larger the savings. Over the long term, using frameworks can reduce support and maintenance costs.

Early in its life, a framework will probably require routine maintenance to fix bugs and respond to client requests. Over time, even the best framework will probably need to be updated to support changing requirements.

Part 2, which will appear in the May 1995 issue, will focus on developing frameworks. For the latest information on Taligent, explore the World Wide Web page //http:www.taligent.com.

Taligent Standards

Early on, Taligent realized the need for common coding standards throughout the company. The chief architect began compiling a list of do's and don'ts that were required reading for all new engineers. This compilation has evolved into *Taligent's Guide to Designing Programs: Well-Mannered Object-Oriented Design in C++*, published by Addison-Wesley. This book, which contains the guidelines followed by the Taligent engineers in the development of the Taligent CommonPoint application system, provides an excellent basis for your own coding standards.

Reuse does not just happen; your organization must actively support and encourage it.

Recommended Reading

The following references include standard object-oriented design references, new publications, and articles about frameworks from a variety of periodicals.

- Beck, Kent. "Patterns and Software Development." *Dr. Dobb's Journal* 19, no. 2 (February 1994): 18.
- Beck, Kent and Johnson, Ralph. "Patterns Generate Architectures," *European Conference on Object-Oriented Programming* (1994).
- Birrer, Andreas and Eggenschwiler, Thomas. "Frameworks in the Financial Engineering Domain: An Experience Report," European Conference on Object-Oriented Programming (1993): 21-35.
- Booch, Grady. "Designing an Application Framework," *Dr. Dobb's Journal* 19, no. 2 (February 1994): 24.
- Booch, Grady. *Object-Oriented Analysis and Design With Applications*. Redwood City, CA: Benjamin/Cummings, 1994.
- Campbell, Roy; Islam, Nayeem; Raila, David; and Madany, Peter. "Designing and Implementing 'CHOICES': an Object-Oriented System in C++," *Communications of the ACM* 36, no. 9 (September 1993): 117.
- Coad, Peter. "Object-Oriented Patterns," *Communications of the ACM* 35, no. 9 (1992): 152.

- Eggenschwiler, Thomas and Gamma, Erich. "ET++ SwapsManager: Using Object Technology in the Financial Engineering Domain," *OOPSLA '92 Conference Proceedings*, ACM SIG Notices 27, no. 10 (1992): 166.
- *Frameworks: The Journal of Software Development Using Object Technology*. Software Frameworks Association.
- Gamma, Erich; Helm, Richard; Johnson, Ralph; and Vlissades, John. "Design Patterns: Abstraction and Reuse of Object-Oriented Design," *European Conference on Object-Oriented Programming* (1993): 406-431.
- Gamma, Erich; Helm, Richard; Johnson, Ralph; and Vlissades, John. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Forthcoming.
- Goldstein, Neal, and Jeff Alger. *Developing Object-Oriented Software for the Macintosh*. Reading, MA: Addison-Wesley, 1992.
- Johnson, Ralph. "How to Design Frameworks," *OOPSLA '93 Tutorial Notes*, 1993.
- Mallory, Jim. "TI Software Speeds Semiconductor Production," *Newsbytes NEW07200011* (July 1993).
- Nelson, Carl. "A Forum for Fitting the Task," *IEEE Computer* 27, no. 3 (March 1994): 104.
- "Semiconductor Industry: New Advanced C.M. software from Texas Instruments Expected to Revolutionize Semiconductor Manufacturing," *EDGE: Work-Group Computing Report* 4, no.166 (July 1993): 22.
- Shelton, Robert. "The Distributed Enterprise," *The Distributed Computing Monitor* 8, no. 10 (October 1993): 3.
- Stroustrup, Bjarne. *The C++ Programming Language*. 2d ed. Reading, MA: Addison-Wesley, 1991.
- *Taligent's Guide to Designing Programs: Well-Mannered Object-Oriented Design in C++*. Addison-Wesley, 1994.
- Wilson, Dave. "Designing Object-Oriented Frameworks." Personal Concepts, Palo Alto, CA 1994.
- Wong, William. *Plug & Play Programming, An Object-Oriented Construction Kit*. M&T Books, 1993.
- Taligent White Papers
  A Study of America's Top Corporate Innovators, Taligent, Inc., 1992.
  Lessons Learned from Early Adopters of Object Technology, Taligent, Inc., 1993.
  Driving Innovation with Technology: The Intelligent Use of Objects, Taligent, Inc., 1993.
  Leveraging Object-Oriented Frameworks, Taligent, Inc., 1993.

Deborah Adair, Taligent, Inc., 10201 North De Anza Boulevard, Cupertino, CA 95014-2233. 1-800-288-5545. Ms. Adair is a staff technical writer in the Taligent Technical Communications Department. She received her BS in Scientific and Technical Communications from the University of Washington.