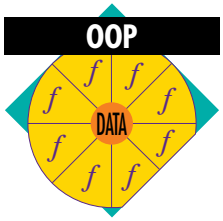
Building Object-Oriented Frameworks

Part 2



By Deborah Adair

By using frameworks, 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 (see the February 1995 issue) described different types of frameworks, how they are used, and addressed organizational concerns that impact framework development. Part 2 outlines a process for identifying and designing frameworks.

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.

Developing Frameworks

The first step in developing a framework is to analyze your problem domain and identify the frameworks you need. Steps for each framework are to identify the primary abstractions, design how clients[1] interact with the framework, and implement, test, and refine the design.

To illustrate these steps, this section describes a small program and one framework that could be used to build it. The program is a data-visualization tool called StockBrowser, which allows users to browse stock histories and display the information graphically. This program enables users to select different stocks and view the price trading volume graphically. The stock information can be displayed as daily, weekly, or monthly values, showing users information such as a stock price on a particular day, or which month had the heaviest volume of trading.

Analyzing the Problem Domain

Once you have identified the problem domain, divide the problem into a collection of frameworks that can be used to build a solution. If the problem maps to a process, describe the process from the user's perspective. For each framework, identify the process it models. Once you have outlined the process, you can identify the necessary abstractions.

Identifying Frameworks

Frameworks can be thought of as abstractions of possible solutions to problems. You can often identify opportunities for new frameworks by analyzing existing solutions.

To determine what frameworks you need, consider families of applications rather than individual programs.

Deborah Adair

[1] 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.

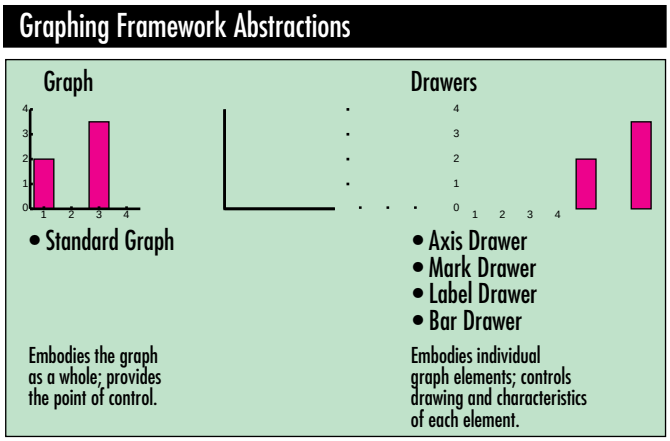


Figure 1. Graphing framework abstractions

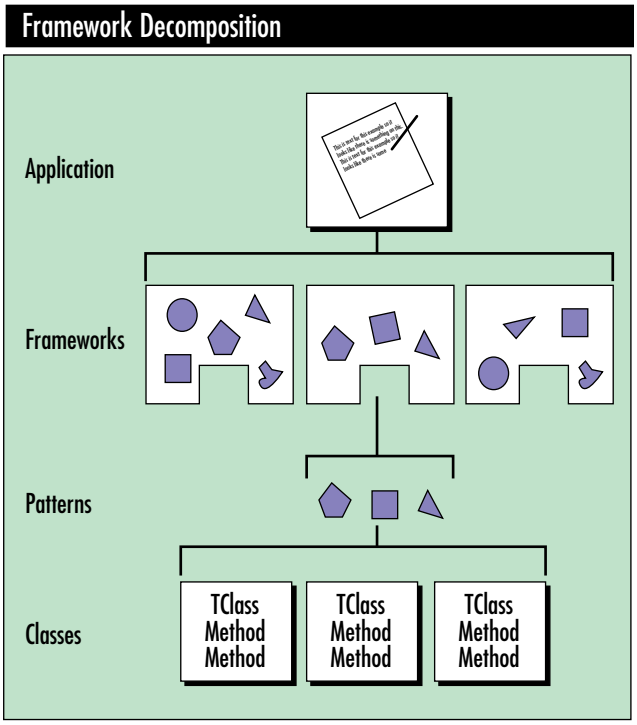


Figure 2. Framework decomposition

- Look for software solutions that you build repeatedly, particularly in key business areas.
- Identify what the solutions have in common and what is unique to each program.

The common pieces—the parts that are constant across programs—become the foundation for your frameworks. Factor these pieces into small, focused frameworks.

When a suite of applications solves similar problems, there is often an opportunity for developing a framework. Potential frameworks can be found in real-world models, processes performed by end users, and source code for current software solutions.

One framework that could be used to build the StockBrowser program is a 2-D graphing framework. The graphing framework would provide support for drawing and labeling the axes and for plotting the data. This type of framework could be used in a wide variety of data-visualization applications—from executive information systems to scientific visualization programs.

Identifying Abstractions

Identify the abstractions your clients need to describe their problems and then provide the logic for producing a valid solution with those abstractions.

The easiest way to identify the abstractions is by using a bottom-up approach—start by examining existing solutions. Analyze the data structures and algorithms, then organize the abstractions. Always identify the objects before you map out the class hierarchy and dependencies.

If you are familiar with the problem domain, you can draw from your past experience and former designs to identify abstractions and begin designing your framework. If you are not an expert in the problem domain, or have not developed any applications for it, examine applications written by others and consider writing an application in the domain. Then factor out the common pieces and identify the abstractions.

There are two primary abstractions for the graphing frameworks: the graph itself and drawers for the graph. Separating the drawing information from the graph enables this simple framework to be extended. New graph elements can be added by providing new types of drawers. Figure 1 shows a graphing framework abstraction.

Designing the Framework

If the framework models a process, determine which steps in the process are performed by the framework, and which steps the client performs. As you design how the framework will work, you might also discover that you can break down the framework into a collection of recurring design patterns, similar to the way you decompose the initial problem into a set of frameworks, as shown in Figure 2.

Each design pattern is a micro-architecture for a recurring element. Patterns can represent generic software elements or elements that are particular to a problem domain. When you design a

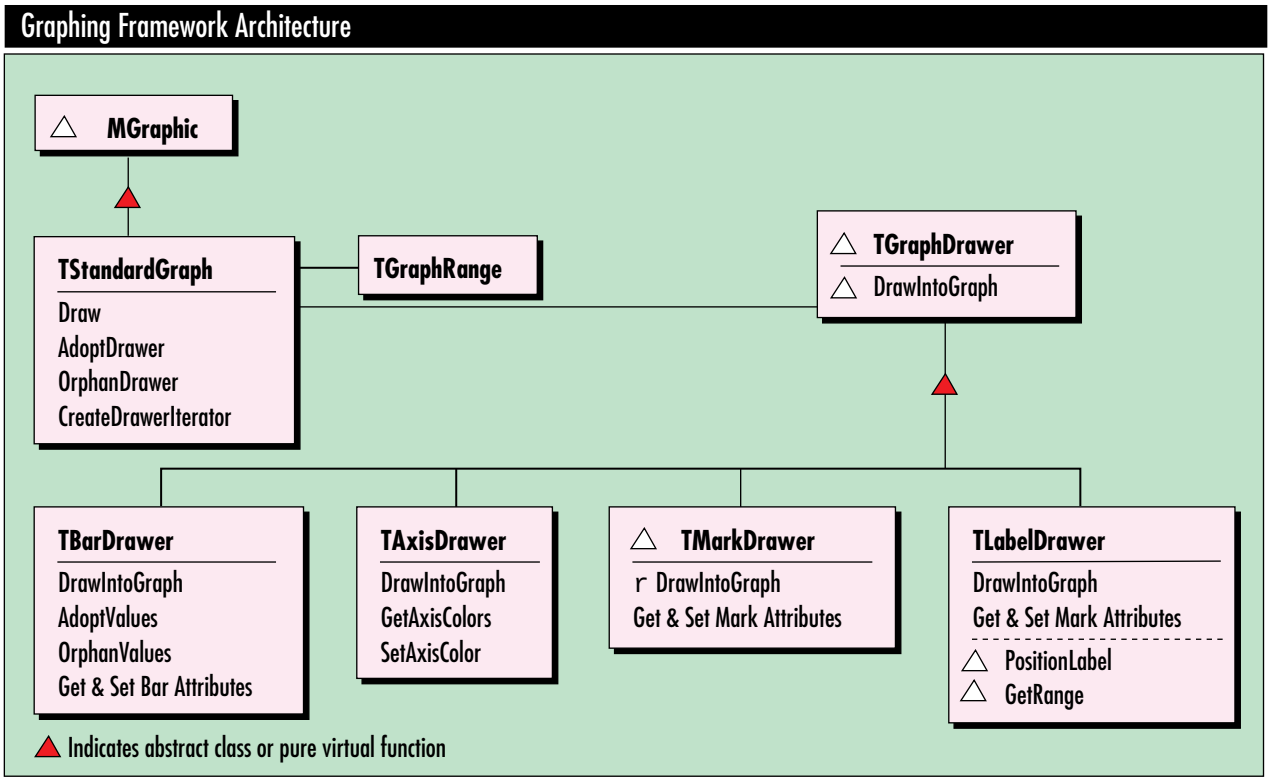


Figure 3. Graphing framework architecture

framework, look for recurring patterns that can be applied to other problems.

The graphing framework supports a simple process—drawing a 2-D graph. The client provides the data for the graph and tells the graph to draw, which then tells each of its elements to draw. The final appearance of the graph is determined by the data provided by the client and the characteristics of the drawers.

On one level, the graphing framework could be described as a manager-driven framework—where the graph functions as the manager. Most framework actions are triggered by telling the graph to draw.

Designing Client-Framework Interactions

In your framework design, focus on how the client interacts with the framework—what classes and member functions the client uses. Look for ways to reduce the amount of code that clients must write by doing the following:

- Provide concrete implementations that can be used directly.
- Minimize the number of classes that must be derived.
- Minimize the number of member functions that must be overridden.

Another consideration is how to support customization. Customization is a two-edged sword—you want flexibility, but you also want to maintain the focus of the framework and minimize the complexity for the client. A completely flexible framework will be difficult for your clients to learn and difficult for you to support.

One approach is to build a very flexible, general framework from which you derive additional frameworks for narrower problem domains. These additional frameworks provide the default behavior and built-in functionality, while the general framework provides the flexibility.

As you design the framework, consider how the design can help communicate the use of the framework. Class names, function names, and how you use pure virtual functions can all provide cues for using the framework.

You also need to determine how the framework classes and member functions interact with client code, such as what objects are created when the client calls framework functions, and when client overrides are called by the framework.

As with most frameworks, the graphing framework provides both features that the client can

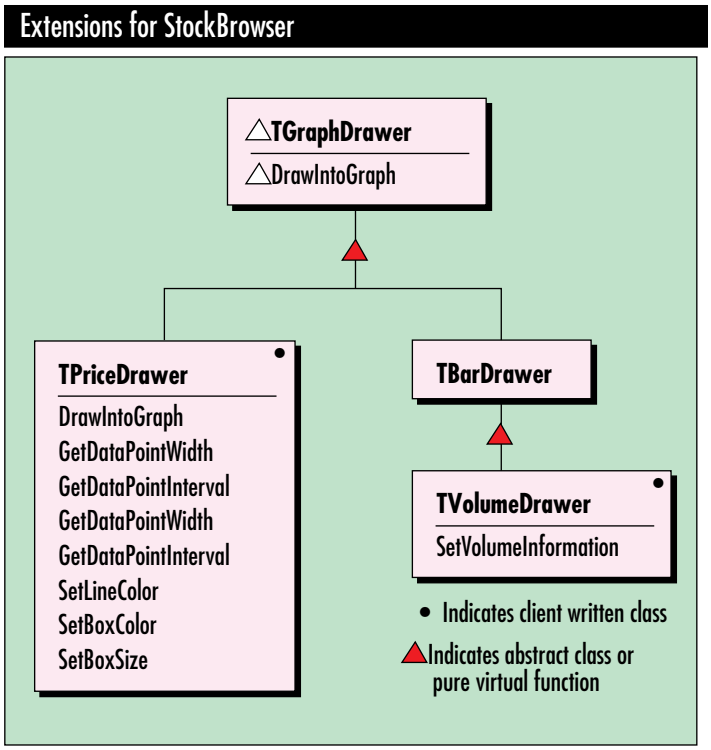


Figure 4. StockBrowser extensions

```
TGraphRange xAxisRange (0, 53);
TGraphRange yAxisRange (0, 100);
TGPoint axisLengths (530, 100);
TStandardGraph graph(xAxisRange,
     yAxisRange, axisLengths);
.
.
GraphValue dataPointWidth = 6;
GraphValue dataPointInterval = 1;
TPriceDrawer* priceDrawer =
    new TPriceDrawer(dataPointWidth,
    dataPointInterval, *data);
priceDrawer->SetLineColor(
    TRGBColor(.75, .2, .75));
priceDrawer->SetBoxColor(
    TRGBColor(0, 0, 0));
graph.AdoptDrawer(priceDrawer);
.
.
graph.Draw(*port);
```

Figure 5. Code for displaying stock data

use directly and an underlying structure that can be extended by deriving new classes. Even this simple framework has both data-driven characteristics and architecture-driven characteristics.

To use the framework, a client constructs the necessary drawers and a standard graph. When the client calls `TStandardGraph::Draw`, then `DrawIntoGraph` is called for each drawer, and the graph and all of its elements are drawn. To change the appearance of the graph, clients can call functions such as `TAxisDrawer::SetAxisColor`.

To extend the framework, clients can derive new drawers from `TGraphDrawer` or one of its subclasses. It is also possible to derive new graph types from `TStandardGraph`—for example, a new type of graph that supports buffered drawing (see Figure 3).

Figure 4 shows how two new drawers were added to the StockBrowser program: `TPriceDrawer` and `TVolumeDrawer`.

`TPriceDrawer` charts stock prices, displaying high, low, and close prices for a particular stock. This class is derived directly from `TGraphDrawer` and from `MStockHolder`, which holds a collection of daily stock data. `TPriceDrawer` overrides `TGraphDrawer::DrawIntoGraph`, and provides functions for manipulating the data points and attributes.

`TVolumeDrawer` charts stock trading volumes in a bar chart. This class is derived from `TBarDrawer`, a generic 2-D bar-chart element, and `MStockHolder`. `TVolumeDrawer` adds the function `SetVolumeInformation` to get the stock volume information to display.

To display the stock data, StockBrowser creates drawers for the graph elements, specifies attributes for the elements, and calls `TStandardGraph::Draw`. The graphing framework does the rest. Figure 5 shows the code for displaying stock data.

Refining the Framework

Building a framework is an iterative process. Beginning with your initial design, you should work with your clients to determine how the framework can be improved—implement features, test them, and verify them with your clients. During this process, you will reanalyze the problem domain and refine your design based on testing, client feedback, and your own insights.

As your framework matures, you will probably find more features to add, and possibly identify opportunities for entirely new frameworks or frameworks that support a particular subset of the problem domain.

Look for additional ways to make your clients' tasks easier. In some cases, it makes sense to provide special tools with your frameworks. Code

generators, CASE tools, and Graphical User Interface (GUI) builders can make programming with frameworks easier, just as they do for traditional software development.

Do not let your frameworks become too large; keep looking for ways to break them down into small, focused frameworks. If they are designed to interoperate, small frameworks are more flexible and can be reused more often.

The nature of an iterative process makes it difficult to determine when a framework is finished. A simple framework requires fewer iterations than more complex frameworks—another advantage of developing smaller, focused frameworks. Until you actually release the framework, you will not gain any of the benefits.

Although prototyping is not unique to framework development, it is very useful. A common approach is to implement a framework that applies to a specific subset of a larger problem domain, and then rework it to support more general cases. The ET++ SwapsManager project is one example of this approach.

After developing a general framework that provides a strong architectural base, you can derive additional frameworks that apply to particular problem sets. The overall framework provides generalized components and constraints to which the derived frameworks conform. Derived frameworks introduce additional components and constraints that support more specific solutions.

Derived frameworks are another method of providing default behavior for your clients. If your framework consists of several abstract classes, you might want to create one or more derived frameworks that provide concrete implementations and additional built-in functionality.

The document framework architecture in the Taligent system supports the creation of most components and documents that will be seen by end users. This set of frameworks was designed to be flexible and allow the construction of a wide range of programs. Derived frameworks support a narrower set of applications. For example, a graphical editing framework simplifies creating programs that manipulate graphical data. Many standard graphics editing functions such as move, scale, rotate, and group are built into this derived framework.

Documenting the Framework

Code comments and documentation are a necessary part of any programming project, but they are especially important if you develop frameworks. Other developers must understand your framework or they will not use it.

Learning how to use another developer's framework can be difficult. As a framework developer, you must provide enough information for your clients to understand how to use the framework for producing the solution they want.

Make it clear what classes can be used directly, what classes must be instantiated, and what classes must be overridden. Clients are interested in solving particular problems—not the details of the framework implementation. It is best to provide a variety of documentation including sample programs, diagrams of the framework architecture, descriptions of the framework, and descriptions of how to use the framework.

At a minimum, provide sample programs. In developing and testing your framework, you have to develop applications that use your framework; often these can provide a foundation set of samples. Try to provide a variety of samples that demonstrate how to use the framework in different contexts—a well-rounded set of sample programs is invaluable to clients learning your framework. Consider designing the sample programs so that they show a progression of the architectural features of the framework.

After developing a general framework that provides a strong architectural base, you can derive additional frameworks that apply to particular problem sets.

Managing Change

Frameworks evolve, especially as your understanding of the problem domain and number of clients grows. However, once you release a framework for client use, you should limit the changes—a constantly changing framework is difficult, if not impossible, to use. As a rule, fix bugs immediately, add new features occasionally, and change interfaces as infrequently as possible.

During framework development, changes happen much more frequently. Once clients have been using a framework, you might still need to make changes that impact their work.

When you update a framework, minimize the impact on your clients. It is better to add new classes instead of changing the existing class hierarchy, or to add new functions instead of changing or removing existing functions.

Give your clients advance notice of framework updates and allow time for them to adapt to the changes. One approach is to add the new and changed classes in an interim release and flag the old ones with obsolete warnings. This warns your clients before their code is broken that they are using classes that are going to change.

Maximizing Framework Benefits

For you and your clients to get the maximum benefit from a framework, the framework should have the following characteristics:

- Provide as much built-in functionality as possible
- Minimize the potential for client errors
- Follow standard design and coding guidelines
- Be extensible
- Be portable

Guidelines for Developing Frameworks

Here are some simple guidelines for developing frameworks:

- Derive frameworks from existing problems and solutions.
- Develop small, focused frameworks.
- Build frameworks using an iterative process driven by client participation and prototyping.
- Treat frameworks as products—provide documentation and support, and plan for distribution and maintenance.

As you develop a framework, you make many small iterations on the design and initial implementation. As the framework matures and your clients begin to rely on it, the changes become less frequent and you move into a much larger maintenance cycle. You can divide this process into four general tasks.

1. **Identify and characterize the problem domain**
   - Outline the process
   - Examine existing solutions
   - Identify key abstractions
   - Identify what parts of the process the framework will perform
   - Solicit input from clients and refine your approach
2. **Define the architecture and design**
   - Focus on how clients interact with the framework:
     - What classes does the client instantiate?
     - What classes does the client derive?
     - What functions does the client call?
   - Apply recurring design patterns
   - Solicit input from clients and refine your approach
3. **Implement the framework**
   - Implement the core classes
   - Test the framework
   - Ask your clients to test the framework
   - Iterate to refine the design and add features
4. **Deploy the framework**
   - Provide documentation in the form of diagrams, recipes, and especially sample programs
   - Establish a process for distribution
   - Provide technical support for clients
   - Maintain and update the framework

Developing Your Own Frameworks

Now that you have an understanding of the overall process and techniques, you can begin to develop your own frameworks. Here are some steps to help you get started:

1. Write a program with an existing framework.
2. Write a different type of program with the same framework.
3. Write a simple framework of your own and test it with small programs.
4. Write a program that combines your framework and an existing framework by using features of both.
5. Develop a small domain-specific framework.
6. Use your framework in more than one project.
7. Try to get other developers to use your framework and develop additional frameworks.

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 a BS in Scientific and Technical Communications from the University of Washington.