Building Object-Oriented Frameworks

Table OF contents

INTRODUCING OBJECT-ORIENTED FRAMEWORKS

Through its use of frameworks, Taligent, Inc. is realizing the full promise of object technology. A framework defines the behavior of a collection of objects, providing an innovative way to reuse both software designs and code.

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 don't 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 may be reused.

Presenting a process for developing frameworks, this paper highlights four important guidelines:

- 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 by providing documentation and support, and by planning for distribution and maintenance

The information is organized into three sections. "Understanding Frameworks" describes different types of frameworks and how they are used. "Maximizing Framework Benefits" addresses organizational concerns that impact framework development. "Developing Frameworks" outlines a process for identifying and designing frameworks. Each section also includes examples of how frameworks are currently being developed and deployed. At the end of this paper, you'll find a brief summary, some tips for starting to build your own frameworks, and a reading list.

While the process and techniques discussed here are ideal for developing in the Taligent® Application Environment, you'll find that they can also be applied to other object-oriented programming projects.

Building Object-Oriented Frameworks 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. Two earlier Taligent white papers, *The Intelligent Use of Objects* and **Leveraging Object-Oriented Frameworks**, provide an introduction to object-oriented design and frameworks for those who are new to the technology.

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.

A framework helps developers provide solutions for problem domains and better maintain those solutions. It provides a well-designed and thought out infrastructure so that when new pieces are created, they can be substituted with minimal impact on the other pieces in the framework.

- Nelson, IEEE Computer, 1994

Understanding Framework Development

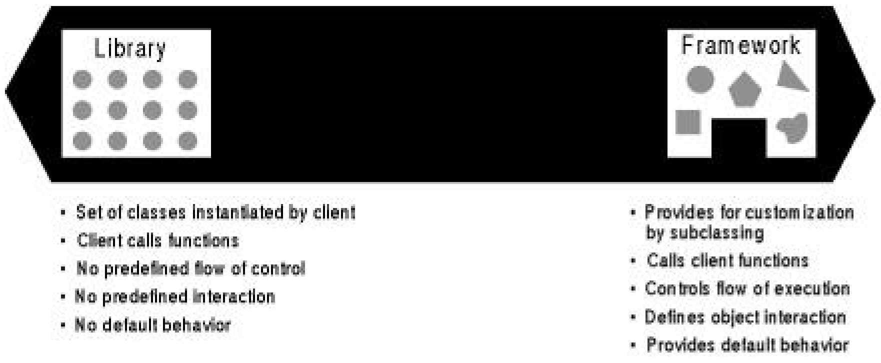
Developing a framework differs from developing a standalone application. A successful framework solves problems that, on the surface, are quite different from the problem that justified its creation.

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

You have to clearly identify the class of problem 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. You can view this as a continuum, with traditional class libraries at one end and sophisticated frameworks at the other:


Library
Framework
• Set of classes instantiated by client
• Client calls functions
• No predefined flow of control
• No predefined interaction
• No default behavior
• Provides for customization by subclassing
• Calls client functions
• Controls flow of execution
• Defines object interaction
• Provides default behavior

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 functions, domain functions, or support functions:

Application frameworks encapsulate expertise applicable to a wide variety of 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 functionality 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 particular client domain. Examples of domain frameworks include: a control systems framework for developing control applications for manu-facturing, securities trading framework, multimedia framework, or data access framework.

Support frameworks provide system-level services, such as file access, distributed computing support, or device drivers. Application developers typically use support frameworks directly or use modifications produced by systems providers. However, even support frameworks can be customized--for example when developing a new file system or device driver.

The Taligent Application Environment frameworks extend across all three categories. These frameworks reduce the amount of code developers have to write and maintain.

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 an independent software vendor, building and using frameworks can increase productivity. The frameworks you build will usually be domain or application frameworks.

Framework Structures

Identifying the high-level structure of a framework can make it easier to describe the behavior of the framework and provides a starting point for designing framework interactions.

For example, some frameworks can be described as manager-driven--a single controlling function triggers most of the 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, for example, generally uses a manager object to take input events from the user and distribute them to the other objects in the framework.

Another categorization is based on how a framework is used--whether you derive new classes or instantiate and combine existing classes. This distinction is sometimes referred to as architecture-driven versus data-driven, or inheritance-focused versus composition-focused.

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 clients to write a substantial amount of code to produce interesting behavior. 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 both use the built-in functionality and modify or extend 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.

Take 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. (See "Using and Extending a Framework" below.)

Using and Extending a Framework
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

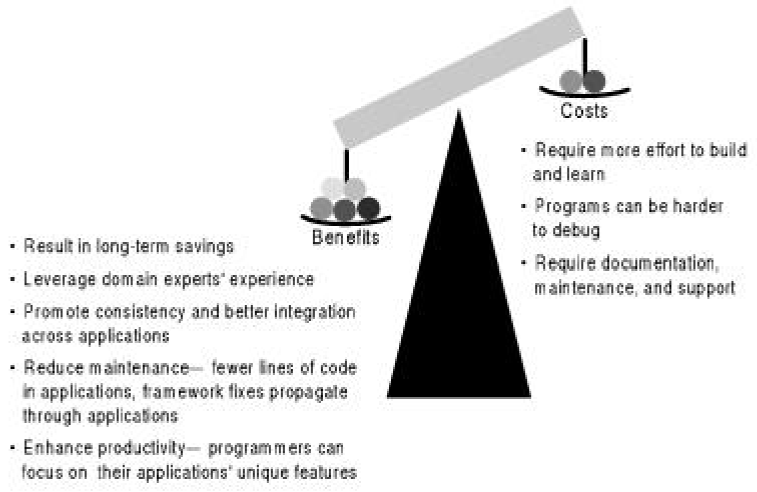
```
class TXMarkContent :
public TboxContent{
public:
virtual void DrawContents(TGrafPort&port)
};
void
TxMarkContent::DrawContents
(TGrafPort&port
{
TLine line1(TGPoint(0,0), TGPoint(10,10)
TLine line2(TGPoint(10,0), TGPoint(0,10)
line1.Draw(port);
line2.Draw(port);
```

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:

- Framework designers need more time to create a framework than a procedural library.
- Clients need more time to learn a framework than a procedural library.

For framework development to be successful, it must be supported by your team's processes and organization.


Costs
• Require more effort to build and learn
• Programs can be harder to debug
• Require documentation, maintenance, and support
Benefits
• Result in long-term savings
• Leverage domain experts' experience
• Promote consistency and better integration across applications
• Reduce maintenance— fewer lines of code in applications, framework fixes propagate through applications
• Enhance productivity— programmers can focus on their applications' unique features

---

The most profoundly elegant framework will never be reused unless the cost of understanding it and then using its abstractions is lower than the programmer's perceived cost of writing them from scratch.

-Booch, Dr. Dobb's Journal, 1994

Combining Efforts

Under contract from SEMATECH, the U.S. chip manufacturing consortium, Texas Instruments Corporation (TI) developed a Computer-Integrated Manufacturing (CIM) framework based on their CIM domain object model. The goal was to develop and implement an industry standard software framework for CIM that could be leveraged to increase U.S. competitiveness in semiconductor manufacturing.

This project was driven by the need to reduce cycle time and cost, and increase flexibility in the production process. The long term goal is a uniform production operation. Providing a framework that supports the core business model for a fab will allow suppliers to provide custom plug and play applications that extend the model.

After developing and deploying a prototype system in a limited production mini-fab, the framework and applications were scaled up for full production. In July `93, TI announced the commercial version, called WORKS.

Sharing the cost and benefits of developing frameworks across companies is becoming popular. As seen with the work sponsored by the SEMATECH consortium, it can also be very successful. Cooperation between organizations through trade groups, informal consortia, and standards groups helps leverage existing experience.

In some cases, even competitors are pooling their resources. For example, the Petrochemical Open Systems Consortium (POSC) was formed by oil and gas companies who realized that their competitive strengths lay in core business areas, not computing technology. As a result, they are cooperatively developing a framework that will allow third party vendors to build custom applications to a standard business model.

Similarly, three Canadian natural gas utilities are working together to build a next generation customer information system. The utilities see savings in four areas: replacement of their legacy system, development cost reduction, reduced maintenance, and cost recovery from the resale of their applications and business framework.

- EDGE: *Work-Group Computing Report,* no. 1003

Managing Dependencies

Projects can often be factored into a number of 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 introduces additional challenges:

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

To alleviate these problems, you can:

- 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 framework, 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 as well.

This same approach can be used to isolate platform-dependent code, or code that accesses a particular applications 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 nonframework services.

Designing Successful Frameworks

To be successful, you should design your frameworks to be:

- 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 demonstrate 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 you're 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's harder for clients to work around bad abstractions than invent their own. If you don't know how to solve a problem in a reusable way inside your framework, leave it out.

A framework doesn't 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, you should 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 it as clear as possible in both your interfaces and documentation what's required of your clients.
- Isolate platform-dependent code to make it easier to port your framework. Designing for portability reduces the impact porting has on your clients.

Delivering Your Framework as a Product

Even if your framework will only be used internally, you must treat it 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.

Distributing and Promoting Frameworks

To use your frameworks, other developers need to know that they exist and know how to access them. Unless a process is established for providing and distributing frameworks, it is difficult to get developers to use them. Reuse doesn't just happen, your organization has to actively support and encourage it. One way to do this is to adjust the reward structures - 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.

Supporting Frameworks

To realize the benefits frameworks can provide, your organization has to commit resources to support them. 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 bigger the savings. 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.

Taligent Standards

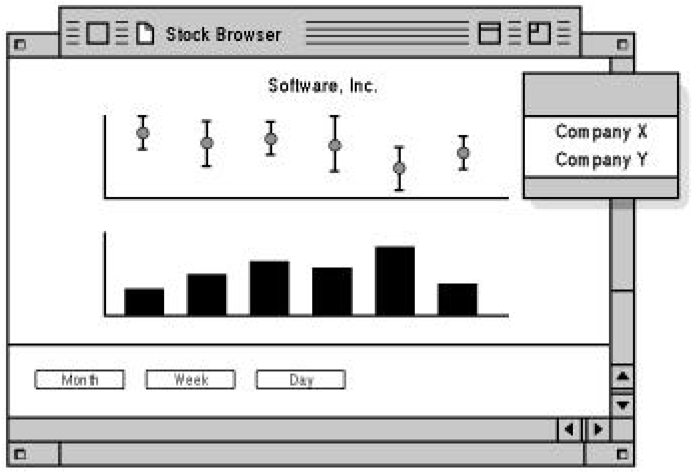
Early on, Taligent realized the need for common coding standards throughout the company. The chief architect began compiling a list of do's and don'ts that were required reading for all new engineers. This compilation has evolved into Taligent`s Guide to Designing Programs: Well-Mannered Object-Oriented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

DEVELOPING FRAMEWORKS

The first step in developing a framework is to analyze your problem domain and identify the frameworks you need. For each framework, you:

- Identify the primary abstractions
- Design how clients interact with the framework
- Implement, test, and refine the design

To illustrate these steps, this section describes a small program and one of the frameworks that could be used to build it. The program is a data-visualization tool called StockBrowser that allows users to browse stock histories and display the information graphically. This program, allows 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 how a stock's price ranged on a particular day or what month had the heaviest volume of trading:



Analyzing the Problem Domain

Once you've identified the problem domain, break the problem down 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've outlined the process, you should be able to 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, think in terms of families of applications rather than individual programs:

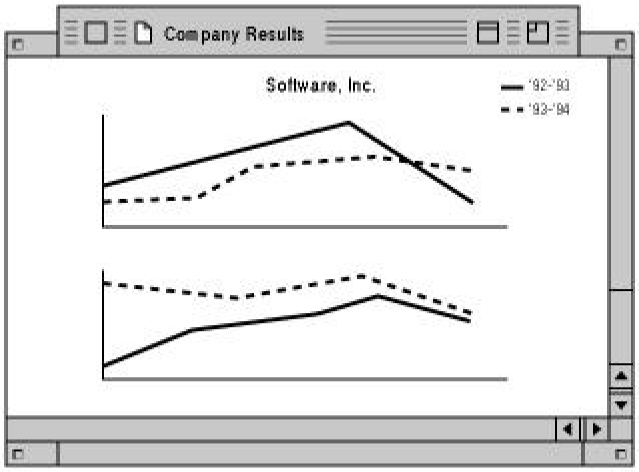
- Look for software solutions that you build repeatedly, particularly in key business areas
- Identify what the solutions have in common and what is unique to each program

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

> Where there's a suite of applications that solve similar problems, there's an opportunity for developing a framework. Look for potential frameworks in:
>
> - Real-world models
> - Processes performed by end users
> - Source code for current software solutions

One of the frameworks that could be used to build this program is a 2-D graphing framework. The graphing framework would provide support for drawing and labeling the axes and for plotting the data.

Such a framework could be used in a wide variety of data-visualization applications--from executive information systems to scientific visualization programs. For example, you might need to build another program that charts sales volume information:



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 with a bottom-up approach--start by examining existing solutions. Analyze the data structures and algorithms and then organize the abstractions. Always identify the objects before you map out the class hierarchy and dependencies.

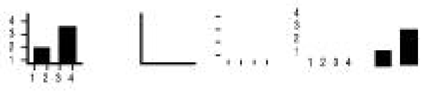
If you are familiar with the problem domain, you can draw from your past experience and former designs to identify abstractions and begin designing your framework.

If you're not an expert in the problem domain, or haven't 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.

For the graphing framework, there are two primary abstractions: the graph itself and drawers for the graph. Separating the drawing information from the graph is what makes it possible to extend this simple framework. New graph elements can be added by providing new types of drawers.

Graphing Framework Abstractions

GRAPH DRAWERS



-Standard Graph - Axis Drawer

- Mark Drawer
- Label Drawer
- Bar Drawer

Embodies the graph Embodies individual
as a whole; graph elements;
provides the point controls drawing
of control. and characteristics
of each element.

Design Patterns

Design patterns identify, name, and abstract common themes in object-oriented design. They capture the intent behind a design by identifying objects, how objects interact, and how responsibilities are distributed among them. They constitute a base of experience for building reusable software, and they act as building blocks from which more complex designs can be built.

Architect Christopher Alexander first introduced the concept of patterns as a tool to encode the knowledge of the design and construction of communities and buildings. Alexander¹s patterns describe recurring elements and rules for how and when to create the patterns. Designers of object-oriented software have begun to embrace this concept of patterns and use it as a language for planning, discussing, and documenting designs.

Just as you perform object decomposition to get from a high level design to an object-oriented implementation, you can decompose a framework into recurring patterns. Some patterns are generic, and some are specific to a problem domain. Each pattern can be characterized by its:

- Preconditions - the patterns that must be satisfied for this pattern to be valid.
- Problem - the problem addressed by the pattern.
- Constraints - the conflicting forces acting on any solution to the problem and the priorities of those constraints.
- Solution - the solution to the problem.

By describing the design of a framework in terms of patterns, you describe both the design and the rationale behind the design.

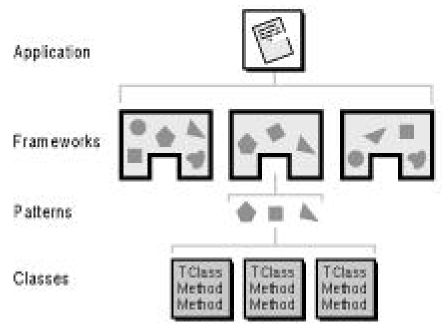
Reusing common patterns opens up an additional level of design reuse, where the implementations vary, but the micro-architectures represented by the patterns still apply.

- - Gamma, Helm, Johnson, and Vlissades, European Conference on Object-Oriented Programming, 1993

Designing the Framework

If the framework models a process, determine which steps in the process the framework performs, and which steps the client performs.

As you begin to design how the framework will work, you might also discover that you can break the framework down into a collection of recurring design patterns, much the way you decompose the initial problem into a set of frameworks.



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

> Good designers know many design patterns and techniques that they know lead to good designs. Applying recurring patterns to the design of a framework is one form of reuse. Using formalized "design patterns" also helps to document the framework, making it easier for clients to understand, use, and extend the framework.
>
> - -Johnson, OOPSLA `93 Conference Proceedings, 1993

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. The graph then tells each of its elements to draw. The final appearance of the graph is determined by the data that the client provides 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 manger. Most of the framework actions are triggered by telling the graph to draw.

Designing the Client-Framework Interactions

In your framework design, focus on how the client interacts with the framework--what classes and member functions does the client use? Look for ways to reduce the amount of code that clients must write:

- 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

One of the things to consider is how to support customization. Customization is a two-edged sword--you want to provide as flexible a framework as possible, but you also want to maintain the focus of the framework and minimize the complexity for the client. If you provide a completely flexible framework, it will be difficult for your clients to learn, and difficult for you to support.

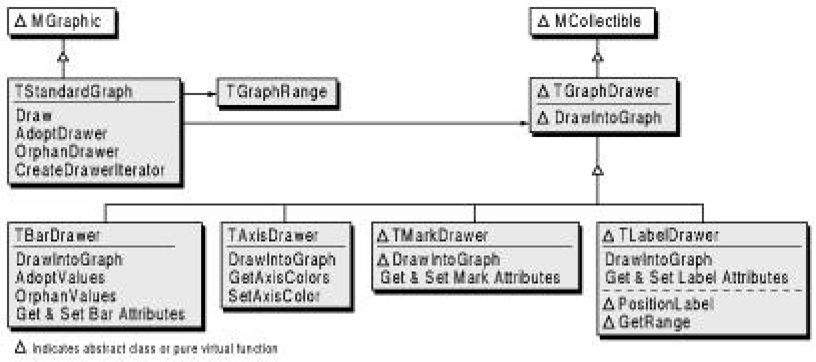
One approach is to build a very flexible, general framework from which you derive additional frameworks for narrower problem domains. These additional frameworks provide the default behavior and built-in functionality, while the general framework provides the flexibility.

As you design the framework, also consider how the design can help communicate how to use the framework. Class names, functions 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:

- What objects are created when the client calls framework functions?
- When are client overrides called by the framework?

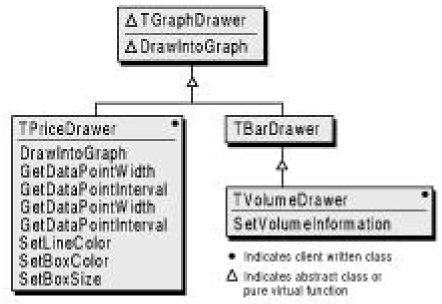
As with most frameworks, the graphing framework provides both features that the client can use directly and an underlying structure that can be extended by deriving new classes--in other words, even this simple framework has both data-driven characteristics and architecture-driven characteristics.



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

To extend the framework, clients derive new drawers from TGraphDrawer or one of its subclasses. It's also possible to derive new graph types from TStandardGraph -for example, a new type of graph that supports buffered drawing.

For the StockBrowser program, two new drawers were added: TPriceDrawer and TVolumeDrawer. (See "StockBrowser Extensions" below.)



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 format. 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.

```
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);
```

Experiences from the ET++ SwapsManager Project

One example of a domain specific framework is the ET++ SwapsManager developed by the Union Bank of Switzerland. This framework was developed as a calculation engine for financial trading software. The ET++ SwapsManager provides a model for swaps trading that demonstrates the advantages of applying framework technology to financial engineering. The developers first analyzed how financial instruments and valuation procedures could be abstracted and then identified their commonalities and interrelationships. Though they had in mind a general framework for financial engineering, for their initial project, they focused on a subset of their problem domain - swaps trading. Their goal was to develop a specific solution that they could expand on. The developers believe that the development of a framework starts with the implementation of a specific solution in a particular domain. You then successively rework the solution to cover a family of applications. Guidelines for generalizing a design:

- Consolidate similar functionality across the system and implement it through a

common abstraction.
- Divide large abstractions into several smaller abstractions. Each of the smaller abstractions should have a small, focused set of responsibilities.
- Implement each variation of an abstraction as an object. This increases the flexibility of the design.
- Use composition instead of inheritance where possible. This reduces the number of classes and reduces the complexity for the framework client.

- - Birrer and Eggenschwiler, European Conference on Object-Oriented Programming, 1993

Refining the Framework

As your framework takes shape, continually look for ways to refine it by adding more default behavior, and additional ways for users to view and interact with the data.

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'll go back and reanalyze the problem domain and refine your design based on testing, client feedback, and your own insights.

As your framework matures, you'll probably find more features to add and identify oppor-tunities for additional frameworks. These might be entirely new frameworks, or frame-works that support a particular subset of the problem domain.

For example, additional features could be added to the graphing framework to make it more flexible. In the current implementation, the label drawer draws only numeric labels, but many graphs use text labels. A better implementation might provide a generic label drawer, with specific derived classes for drawing text labels and numeric labels. It could also be useful to add drawers for other types of graphs, such as line graphs and pie charts. Or perhaps a new framework to support 3-D graphs would be useful.

Also, 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 GUI builders can all make programming with frameworks easier, just as they do for traditional software development.

Don't let your frameworks get too big--keep looking for ways to break them down into small, focused frameworks. If they're designed to interoperate, small frameworks are more flexible and can be reused more often.

For example, if you needed support for creating distributed workflow applications, you might initially design a workflow framework with these features:

- Document management
- Rule-based agents
- Store and forward servers

- Networking

A better solution would be to develop separate frameworks for each major function--for example, a document management framework, an agent framework, a client/server framework, and a networking framework. By breaking down the original workflow framework into a set of small frameworks, the resulting frameworks can be used in other contexts.

It can be difficult to determine when a framework is finished--that's the nature of

an iterative process. A simple framework requires fewer iterations than more complex frameworks--another advantage of deve-loping smaller, focused frameworks. Until you actually release your framework, you won't gain any of the benefits.

Generalizing the Design

The concept of prototyping is not unique to framework development, but 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.

Deriving Frameworks

Once you've developed 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 con-straints 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 is made up of a number of abstract classes, you might want to create one or more derived frameworks that provide concrete implementations and additional built-in functionality.

For example, in the Taligent system, the document framework architecture supports the creation of almost all components and documents that will be seen by end users.

This set of frameworks was designed to be very 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 the creation of 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 when you're developing frameworks. If other developers don't understand your framework, they won't use it.

Learning how to use someone else's framework can be difficult. As a framework developer, you have to provide enough information so that your clients understand how to use the framework to produce 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 - the details of the framework implementation are not important.

It is best to provide a variety of documentation:

- Sample programs
- Diagrams of the framework architecture
- Descriptions of the framework
- Descriptions of how to use the framework

> Examples make frameworks more concrete and make it easier to understand the flow of control.
>
> - - Johnson, OOPSLA `92 Conference Proceedings, 1992

At a minimum, provide sample programs. In the process of 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.

Managing Change

Frameworks evolve--especially as your understanding of the problem domain and number of clients grow. However, once you release a framework for client use, you need to limit the changes--a constantly changing framework is difficult, if not impossible, to use.

As a general rule, you should:

- Fix bugs immediately
- Add new features occasionally
- Change interfaces as infrequently as possible

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

When you do update a framework, minimize the impact on your clients. It's 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 way, your clients are warned before their code is broken that they are using classes that are going to change.

SUMMARIZING THE FRAMEWORK DEVELOPMENT PROCESS

Developing frameworks allows you to solve a problem once and then reuse that solution any time you face a similar problem. After you've captured domain expertise in a framework, you and others can reuse that framework, saving time and money both in program development and in maintenance. To maximize this benefit, develop a collection of frameworks that you can select from to construct each new program.

You can categorize frameworks by the types of problems the frameworks solve or by the framework structure. The frameworks that you build will likely be domain frameworks, or possibly application frameworks.

Maximizing Framework Benefits

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

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

Your goal as a framework designer is to get other developers to use your frameworks.

If they don't understand how your framework works and how to use it, they'll develop their own solution. You must strike a balance between the simplicity of the client interface and the power of the framework.

When in doubt, err on the side of making it simpler for your clients to use the framework, even if it makes implementing the framework more difficult.

Developing Frameworks

When you develop 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

Following these guidelines can help you build frameworks that your clients need, that they can use, and that you can support and maintain.

When you begin to 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:

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

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

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

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

Starting to Develop Your Own Frameworks

Now that you have an understanding of the overall process and techniques, you can begin to develop your own frameworks. To get started, you might:

- Write a program with an existing framework.
- Write a different type of program with the same framework. Write a simple framework of your own and test it with small programs.
- Write a program that combines your framework and an existing framework by

using features of both.

- Develop a small domain-specific framework.
- Use your framework in more than one project.
- Try to get other developers to use your framework and develop additional frameworks.

RECOMMENDED READING

The experiences of others are a great source of information and inspiration--you are encouraged to read about what other groups are doing, as well as publish articles about your own endeavors.

The following references include standard object-oriented design references, new publications, and articles about frameworks from a variety of periodicals. Many of the examples included in this paper are based on information in the articles listed here.

Publications

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, to be published October 1994.

Goldstein, Neal, and Jeff Alger. *Developing Object-Oriented Software for the Macintosh*. Reading, MA: Addison-Wesley, 1992.

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 CIM 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.

Wong, William. *Plug & Play Programming, An Object-Oriented Construction Kit.* M&T Books, 1993.

Seminars

Wilson, Dave. "Designing Object-Oriented Frameworks." Personal Concepts, Palo Alto, CA 1994.

Johnson, Ralph. "How to Design Frameworks." *OOPSLA '93 Tutorial Notes*, 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.

JavaTM is a trademark of Sun Microsystems, Inc.

Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.

Other companies, products, and service names may be trademarks or service marks of others.

Copyright Trademark

Java Education **Java Home**

IBM HOME ORDER EMPLOYMENT CONTACT IBM LEGAL