Architecture Patterns for Business Systems

Lorraine L. Boyd
American Management Systems
4050 Legato Rd.
Fairfax, VA 22003
703-267-5497
email: lorrie_boyd@mail.amsinc.com

Overview

Software architecture for business systems is like the girding of a large building. It satisfies characteristics needed by the occupant, shapes the overall structure, and defines the boundaries of individual rooms. Unlike buildings, however, large scale business systems have contradictory characteristics which can only be met with different structures. It's as if we're constructing a city, not just a single building.

This paper defines some primary architecture characteristics of business systems, and, based on user requirements to meet these characteristics, proposes four engineering models for architectural patterns which together define an overall software architecture.

First, this paper describes seven primary characteristics of business systems. Different business functions within a system have contradictory user requirements for each of these characteristics. The contradiction and tension in these user requirements become forces for the individual engineering model architecture patterns.

Then, this paper presents four engineering model patterns which together meet the main requirements of these characteristics. The engineering models are: Online, Messaging, Batch and Stream IO. Online provides quick response to user interaction, using unpredictable access paths across all the complexity of the business model. Messaging, although possibly initiated by user interaction, provides complex processing with delayed responses using more predictable access paths. Batch is scheduled, has no user interaction, and provides processes which are highly complex, highly predictable and have high throughput demands. Stream-IO is initiated by an event rather than by user interaction or a schedule, and provides apparently immediate processing of individual events.

By delineating and tuning different engineering model architecture patterns to meet different user characteristics of different functions, the overall system will meet all the user requirements. In addition, defining these architecture patterns early in the analysis phase provides a framework for designing individual business functions. Designers can evaluate their particular business function against the system characteristics and determine the best architectural engineering model to use as a framework for their function.

Architecture Characteristics of Business Systems

Like cities having different buildings, large scale systems have different business functions which have different software requirements. These requirements can be categorized by the characteristics listed here. This categorization of characteristics is based on the author's personal experience of 20 years of designing and implementing large scale business systems, as well as the cumulative experience of American Management Systems' thousands of successful systems implementations. A software architecture needs to be framed to meet the often contradictory requirements of these characteristics for each of the different business functions. This paper will compare example business functions within a system against these characteristics. The four engineering models developed together become a pattern language for an overall software architecture which integrates all requirements for these characteristics.

- *User Interaction*. High user interaction means a person is constantly interacting with the function, initiating actions and waiting for results. No user interaction means the function can process independently – no person initiates or waits for the results.

- *Predictability of Access*. Low predictability means the processing and the data accessed cannot be predicted in advance. High predictability means the processing and the data access paths are generally the same with each execution of the function.

- *Object Model Structure*. Complex structure means this function may use the entire complex business object model. Simple structure means this function uses a predictable, simple object structure which is perhaps comparable to a flat file.

- *Throughput*. Low throughput means the total activity volume per second on the function as a whole is small. High throughput means the total activity volume per second on the function as a whole is high.

- *User response time*. High user response time means each individual action in this function must be completed quickly – a person is usually sitting and waiting for an answer. Low user response time means each individual action need not be completed quickly – it might be acceptable if individual answers are not available for several days.

- *Process initiation*. User driven initiation means a user driven event initiates the function. Time driven initiation means the function is scheduled and initiated at a predictable point in time.

- *Distributed processing*. Highly distributed means there are overriding technical reasons for distributing processing for this function among the client, the middleware and the server. Highly centralized means there are overriding technical reasons for keeping the processing of this function centralized.

Pattern Name: Online Engineering Model

Problem:

User interaction is as familiar as Web graphics today, or as 3270 mainframe screens from 20 years ago. It characterizes much of what is written and taught about business computer systems. The user interaction portion of business system is so familiar that we often overlook the variations in requirements which drive the architecture decisions.

How can we determine the highly user iteractive functions of our system and assure our architecture meets these user requirements?

Context:

Consider a business system which supports the day to day operations of a business. This system supports the telephone representative answering customer queries, fills orders, calculates bills, and prices the products created. Each of these functions are considered mission critical to the manager responsible for them. Our system needs to respond to all needs.

When a customer queries, the system must provide an answer within 2-3 seconds. The telephone representative waits for the computer to respond while the customer is on the line. Customer requests are unpredictable, but world class customer support is considered a mission critical business differentiater in the marketplace. As a result, an online user interaction architecture has been developed and tuned to provide broad, flexible access with quick response for individual queries.

Forces:

The customer queries function has the following requirements of each of the primary system characteristics:

Characteristic	Customer Query Requirements
User Interaction	Very highly interactive. User initiates request and waits for answer
Predictability of Access	Not predictable. Requests and access paths are random.
Object Model Structure	Requires processing and accessing across the entire, complex business object model.
Throughput	Relatively low. In the system as a whole relatively few actions will be requested each second.
User Response Time	Relatively high. For each individual request, the user waits for a response and expects one within 2-3 seconds.
Process Initiation	Processes are always initiated by user driven events.
Distributed Processing	Highly distributed. Minimizing the processing at the client greatly reduces overall system cost.

Solution:

Define an online engineering model to support business functions with these characteristics. This architecture pattern will provide a framework for highly user interactive functions with complex and unpredictable access paths. The online model supports immediate response, interactive error recognition, flexible access, and thin clients. Thin clients means minimal processing is done at the client PC, thus reducing the size of each PC needed and therefore significantly reducing the overall hardware costs of the system. Although the response time for an individual action needs to be quick – someone is waiting for an answer, the overall system throughput is relatively low since the user may wait several minutes between each action. Figure 1 illustrates the online engineering model.

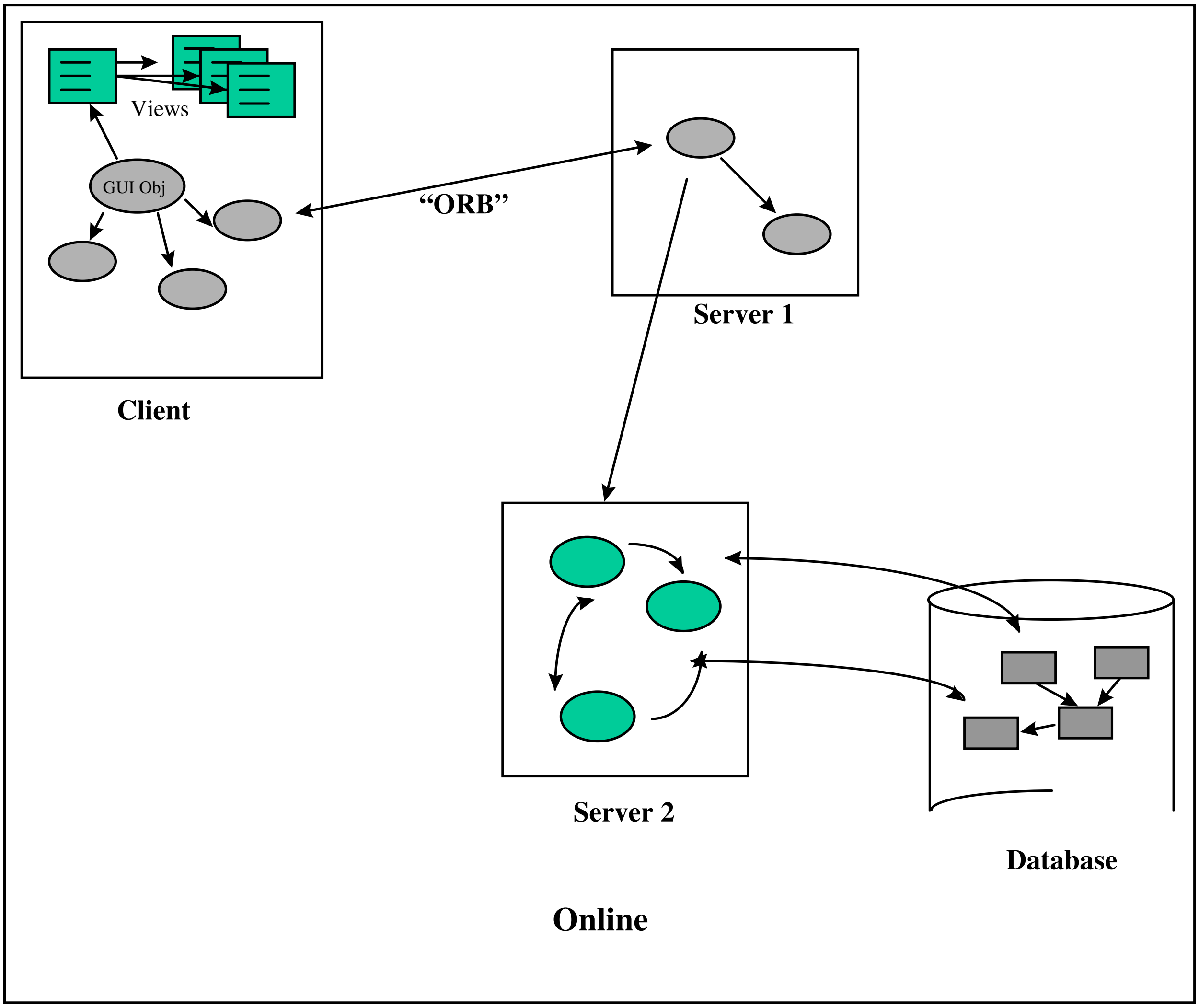

Figure 1

This figure illustrates the online engineering model. All work is highly user interactive, with the GUI screens initiating and responding to the system processing. Since users can click and go practically anywhere in GUI screens, access paths are simply not predictable, and each request may traverse unpredictably different areas of the entire complex object model. Users will expect a quick response to each individual button click, while keystrokes and human work will mean a relatively low overall throughput for the system overall. If using distributed objects, an Object Request Broker (ORB) may be used to communicate between the client and the server. As in Figure 1, multiple servers may be used to further distribute the processing. But, whether or not ORB is used, the processing is highly distributed. Screen or view processing is on the client (views and GUI objects) , and model and business processing is on server(s), and the database tables are further distributed possibly on their own server.

Example:

This pattern is used in all customer query driven actions. In the system discussed in the context section, customers may call the company representative to query for account balances, to request adjustments in their accounts, to change their address, to purchase products, to cancel orders or any number of different requests. In all cases, the customer waits on the phone and expects an immediate response. To support this, the view and GUI objects present the user interface and do some formatting and editing of information entered, while the model and business objects on the servers process the request, and update and fetch the data from tables in the database.

Known Uses and Related Patterns:

The online model is familiar and serves as the basis from which the other engineering models within this pattern language diverge. This model is used in the customer care portion of business systems where service representatives answer queries and provide service to customers. Many architectural patterns, i.e., Model-View-Controller, Pattern Oriented Software Architecture – A System of Patterns (ref 1)and Cookbook for using Model-View-Controller User Interface Paradigm (ref 4); Transactions, Analysis Patterns (ref 2), are approaches for object design within this engineering model. For example, in the Model-View-Controller pattern the Model component encapsulates core data and functionality, the View component displays information to the user, and the Controller component translates events the into service requests for the Model or the View.

Pattern Name: Messaging Engineering Model

Problem:

Some functions in most business systems, although initiated by user interaction, do not have the same requirements as the online engineering model. For example, some functions may be highly complex and require too much processing time for a user to wait for an answer. In addition, the business may not actually require an immediate answer.

How do we identify the functions of our system which are less user interactive and more processing intensive? And how do we develop a software architecture which supports these functions?

Context:

Consider a the operational business system discussed above. All user interactive functions may not be able to be completed in the online requirement of 2-3 seconds.

For example, the processing of some complex orders may require an unpredictably long amount of time – perhaps hours. And although this scheduling is initiated by user interaction, the business understands the complexity and no one expects to wait for an immediate answer. Certainly it is also critical to the business to be able to fill complex orders correctly, even if this has less marketing sizzle.

How can user interactive functions, with both high and low response time requirements, be met by the software architecture? How can the architecture developed provide direction to developers, perhaps junior staff, so that these different requirements are noticed during design, and are met during development?

A separate architectural engineering model is needed to support both types of user interaction.

Forces:

Characteristic	Complex Order Requirements
User Interaction	Partial. Initiated by user interaction, but no interaction after the initial request.
Predictability of Access	Some predictable access. Requests have known parameters, access paths can be determined from these parameters.
Object Model Structure	Requires processing and accessing across the entire, complex business object model.
Throughput	Relatively low. In the system as a whole relatively few actions will be requested each second.

Characteristic	Complex Order Requirements
User Response Time	Medium. No one is waiting for an answer. The user understands complex requests may take hours.
Process Initiation	Processes are always initiated by user driven events.
Distributed Processing	Some distribution. With little user interaction, minimal processing needs to be on the client. Most of the processing can be done on the message oriented middleware layer.

Solution:

Define an messaging engineering model architecture to support complex asyncronous user interactive processing. Separate this engineering model from the basic online architecture.   Make both architecture models available throughout the design process so that developers question their users and determine which architecture is best for each business requirement.

The messaging engineering model will be tuned for user interactive initiated requests, which may have many parameters. Once the request is made, this model processes in the background while the user continues with other work. The processing work is often complicated and the architecture is tuned for relatively low volume throughput, and relatively complicated processing. When processing is finished, this model returns the answer to the user, either online or in a messaging queue.   Figure 2 illustrates the Messaging Engineering Model.

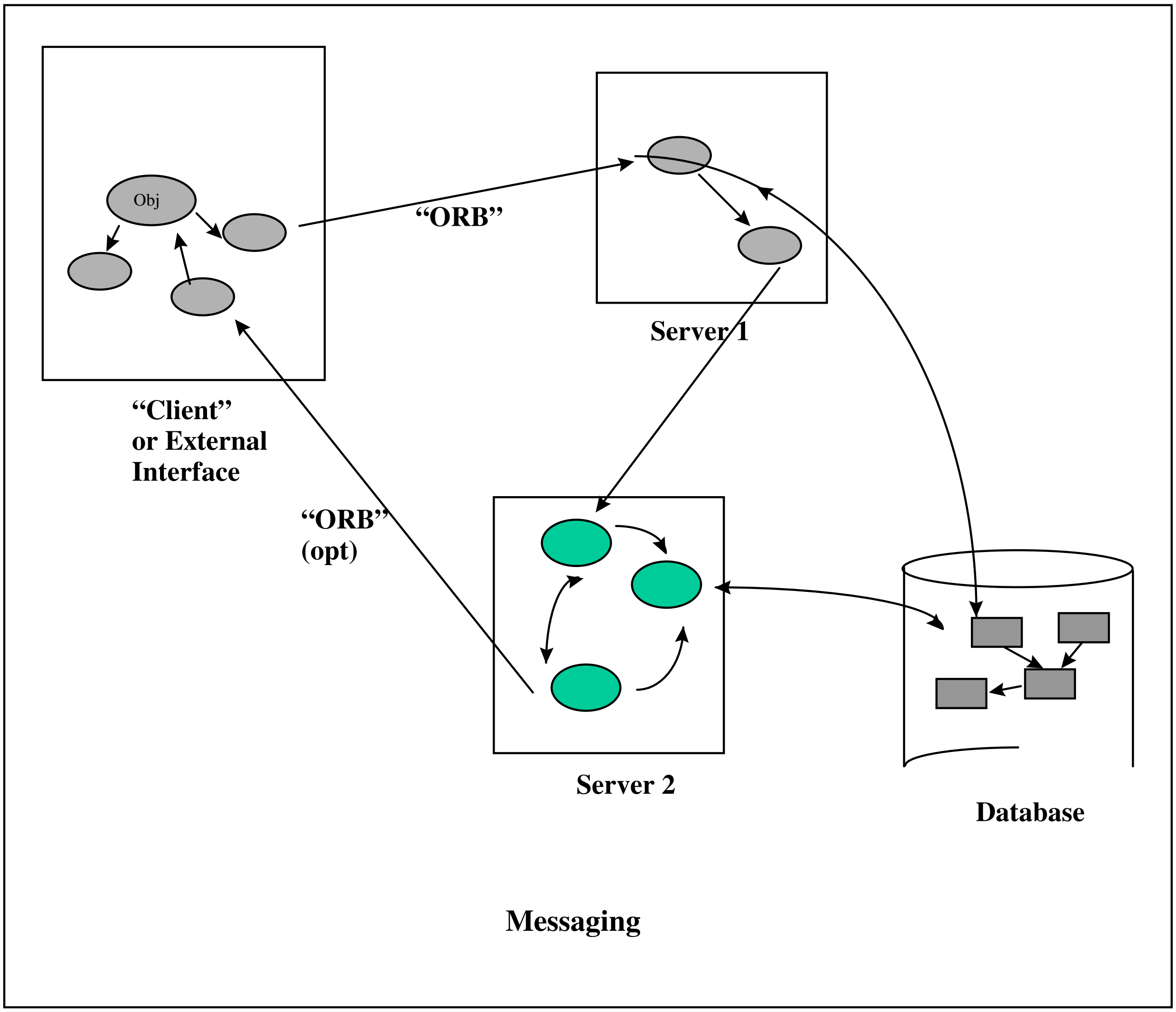

Figure 2

This figure illustrates the messaging engineering model. All work is initiated by the objects on a client, which may be objects managing Views, or which may be external interfaces. The requests are more predictable than online, and requests can be predetermined. The biggest difference between this model and the online model is the lack of user interaction, and the potentially slower response time. Both models use the same underlying business object model and database. The differences are in the distribution and initiation of system set-up classes. The client could be the same as in figure 1, with the same GUI objects. The views, which reside on the client, do not interact with the messaging pattern. In addition, the messaging pattern could be initiated by any external interface. Messaging actions may have their own controller objects (on server 1), but can use the same business objects on server 2 and the same database as in figure 1.

Example

The telecommunications business is changing rapidly. Customer queries can concern adjustments to the price of particular calls, requests for different types of service, or address changes. Each of these involve user interactive, flexible processing. On closer look, however, there are significant differences in the user interactive requirements. It's this closer look which this messaging engineering model supports.

For example, some orders for service are simple. An individual consumer customer requests a change in service, the telephone representative adds the service, the computer system activates the service in the system database and at the switch. The customer is told he or she can now use the requested service.

Some orders for service are very complicated. An international business requests a PBX switch. They do not expect the switch to be installed while they wait. The business requests the switch from their dedicated representative, the representative enters each of the parameters needed to schedule the install, several different companies and switches (each with a different interface to our computer system) may need to be coordinated in order to complete the entire request, the representative does other work while the computer schedules the installation, the computer notifies the representative when the scheduling is completed, the representative calls the business with scheduling choices.

Known Uses and Related Patterns:

This pattern is related to the Online engineering model because it can be initiated by user interaction and has relatively low throughput. It differs in that it defines complex asyncronous processing. Examples of business functions which require this architecture pattern include complex order management in telecommunications, and mass changes in any system with a very large database.

Pattern Name: Batch Engineering Model

Problem

Twenty years ago, before 3270 screens were even invented, business systems ran in batch mode. Someone made a request, the request was held in a transaction queue, the entire queue processed as a batch overnight, the system provided an error report and a list of successfully processed requests the next morning. The traditional transaction process of transaction queue and master file, with processing delayed after the instant of the request, is still a technical approach many legacy systems carry with them.

It is also a model of how some business functions actually operate. Some business functions are initiated by the passage of time, not by user interaction. These functions usually process against very large volumes and require very high throughput. What these functions do is relatively predictable, and often access only a narrow slice of the overall system information. The business does not need, or certainly isn't willing to pay for, immediate response for each individual request.

These particular business functions have no sizzle at all. Although they are necessary for businesses to run, they are often the last pieces designed. Literature, i.e., Relational Database Access Layer", Proceedings PloP 96 (ref 3), about business system architectures often excludes batch, because it is so different. Even testing is different. Batch test scripts need to include integrating online with batch, processing huge volumes, and conceptualizing as a group rather than as an individual request.

Batch functions are rarely discussed in training or in literature, and are often ignored in the design process until an architecture which is only appropriate for online user interactive processing is all that is in place. By the time we notice the boring batch stuff, the team is fully committed to other functions.

So we try to develop batch business functions as a kluge of the online functions. Or we ask the Cobol programmers to do the batch functions without objects.

How do we create a system which defines different requirements for both batch and user interaction; and which supports each of these different requirements; and which provides direction, through the architecture, for development?

Context

Consider the business system mentioned earlier. That system needed to respond to customer queries and it also needed to calculate bills. Calculating bills has some very different business needs from responding to customer queries.

In billing, customers select a date to receive the monthly bill. Then, each day, the system calculates the bills for all customers who have selected that date, updates corporate financials, and directly debits customer bank accounts for any customers with that payment method. There is no user interaction after the billing date is selected. For many businesses, these processes require a very high throughput – thousands of bills may be processed each second -- for relatively predictable, if complicated processing.

These requirements are not the same as online processing. An architecture tuned to support only online processing would not be able to support these functions.

How can both of these needs be met by the system architecture? How can any architecture developed provide direction to developers, perhaps junior staff, so that these requirements are noticed during design, and are met during development?

A separate architectural engineering model is needed to support batch as well as online processing..

Forces

Characteristic	Billing Requirements
User Interaction	None.
Predictability of Access	Highly predictable. Process for calculating bill is known.
Object Model Structure	Requires only a simple subset of the business object model. Subset can be characterized as flat files.
Throughput	High. For each execution very high volumes process per second.
User Response Time	Low. A new bill is only expected once a month.
Process Initiation	Scheduled. Processes are always initiated at a predictable time.
Distributed Processing	None. Processing will be more efficient, and therefore quicker, if all processing occurs on the server.

Solution

Define an batch engineering model architecture. Separate this engineering model from the online architectures (Online Engineering Model and Messaging Engineering Model). Make all models available throughout the design process so that developers question their users and determine which architecture is best for each business requirement.

The batch engineering model will be very different from the online architectures, yet will integrate in the data access or domain model layer. The batch engineering model will provide:

- the ability to initiate work without user interaction, usually triggered by time,
- error processing which holds errors in a separate queue, rather than immediate notification of users,
- performance tuned for very high volume throughput with predictable access
- locking and data handling different from online use to allow concurrency with online
- reporting of audit information after a group of records are processed

- the ability to restart the job in the middle, recognizing and bypassing any records which were already processed
- centralized processing on the server with no client interaction

Figure 3 illustrates the batch engineering model.

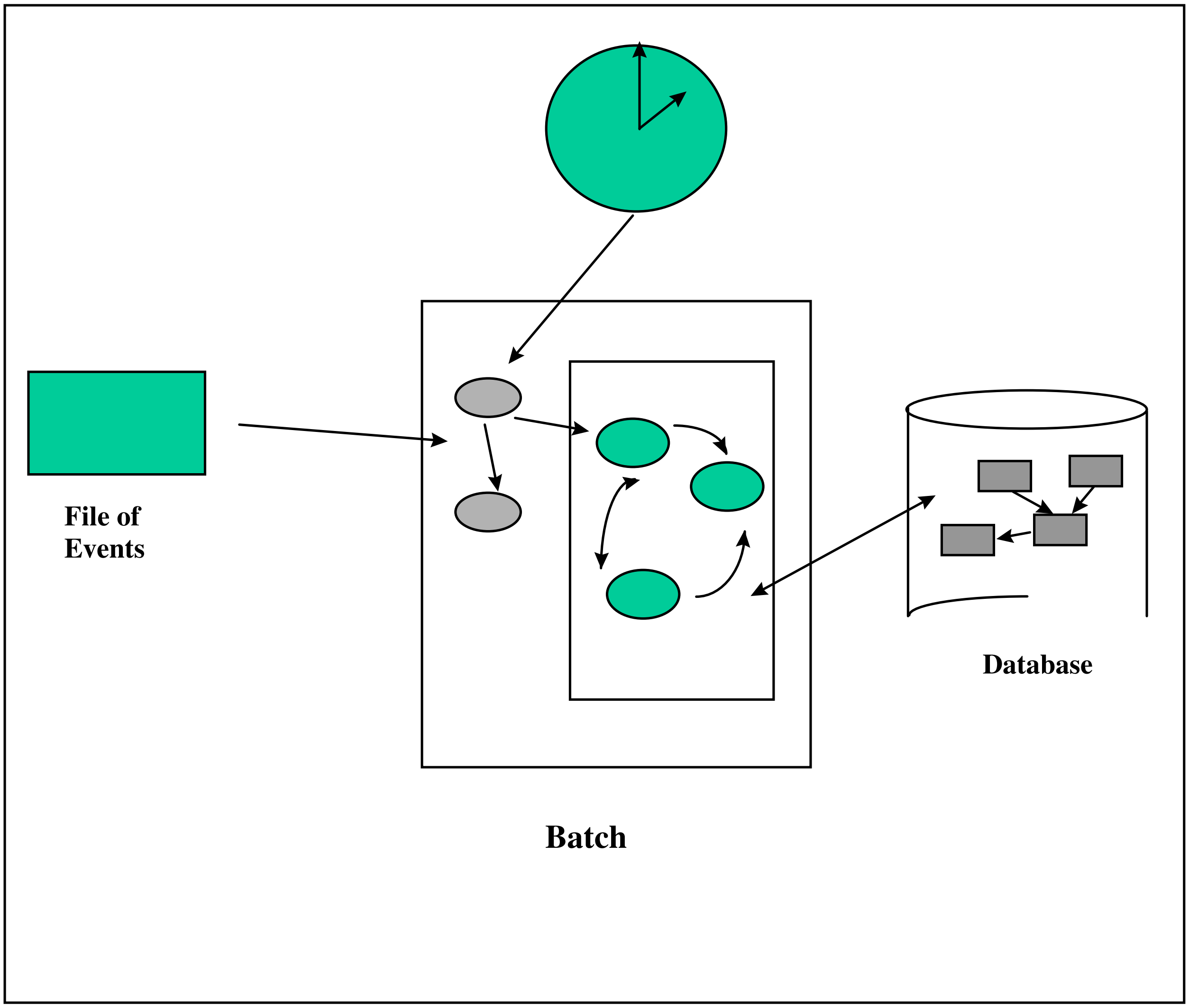

Figure 3

This figure illustrates the batch engineering model. All work is scheduled. In other words, it can be initiated at a point in time. In addition, all processing is centralized on a single platform. This centralized processing provides very high throughput, as volumes for each individual batch function may be very high. These processes are predictable, and what is needed for each 'transaction' in the file of events can be predicted in advance. Note, the batch engineering model uses the same underlying business object model and database as both other models. Again, the differences in the software architecture is in the distribution and initiation of system set-up classes. The file of events is an external file of transactions which the batch system needs to process as a group.

Example

In nearly every major business system, the primary example of batch processing is Billing. Customers want to receive their bill on a regular, predictable time each month. To meet this need, on the last day of each month, at midnight, the system run bills and sends to all customers.

For some businesses, this is the most complicated processing in the entire system. There may be multiple different pricing mechanisms, and multiple products. At the same time, accurate and timely billing is clearly also critical to the success of the business.

Defining and using a batch engineering model provides a framework for the technical solution which allows developers to focus their time on defining exactly how the bills need to be calculated. In addition, providing the framework as an architecture engineering model assures that these functions will be addressed appropriately and early.

Known uses and related patterns

This architecture pattern is the opposite of the Online Engineering Model – the user requirements of the system characteristics for these two model patterns are at opposite ends of the scale. Yet, the batch model is appropriate for many of the operational functions of business. Primary examples include: Calculate billing and Calculate payroll. In addition, this model is used to manage many external system interfaces where files periodically are sent to or received from external systems.

Pattern Name: Stream IO Engineering Model

Problem

Just as the old standby online architecture did not meet all user interactive needs and required the messaging engineering model for some functions, so a single batch engineering model isn't able to do everything. The batch needs are equally diverse.

Businesses and customers are demanding more and more immediate feedback. Many functions still have technical reasons for running in batch mode – for example high throughput and predictable access path. However, the business of today defines different functional requirements from those defined above for the Batch Engineering Model.

Some business functions, which are not user interactive, are driven by an event rather than the passage of time. To further complicate the difference, the event may appear to happen continuously. To the business, this function is always going on, and has no discrete beginning and end.

These functions are neither online, nor batch. They share characteristics of both, and will not meet the business needs if forced into either engineering model.

How do we create a system which defines different requirements for these continuous processes; and which supports each of these different requirements; and which provides direction, through the architecture, for development?

Context

Consider the business system mentioned in each of the other engineering models. That system needed to respond to customer queries, send bills, and price the products created. In a large business, particularly a service oriented business, products – such as phone calls – can happen constantly.

The technology of legacy systems may have delayed the pricing of products until a batch job ran. Today, however, businesses are demanding to know the price of individual products immediately. While the business may want to pay employees, or bill customers, on a monthly basis; there may be significant business benefit to knowing the costs of products immediately.

This pricing still may not involve any user interaction. Pricing is initiated by an event, such as a phone call. No one needs to be notified each time an individual call is priced. But the result may need to be available to other user interactive queries immediately.

These functions require are a little bit of everything. They are often have the highest volume throughput of the entire system. They run on their own, like messaging. They have predictable access paths and very high throughput, like batch. Their results need to be available immediately, like online.

How can the design of these functions be answered by the system architecture? How can the sytem architecture respond to these critical business needs, as well as the other conflicting needs?

A separate architectural engineering model is needed to support stream IO processing..

Forces

Characteristic	Product Pricing Requirements
User Interaction	None.
Predictability of Access	Highly predictable. Process for pricing individual products is known.
Object Model Structure	Requires only a simple subset of the business object model. Subset can be characterized as flat files.
Throughput	High. For each execution very high volumes process per second.
User Response Time	Medium. Users may wish to query on results quickly.
Process Initiation	Event driven. Processes are always initiated by an event, although a user at a view does not create the event.
Distributed Processing	Little. Maybe initiated at the middleware, but processing will be more efficient, and therefore quicker, if most of it occurs on the server.

Solution

Define a stream I-O engineering model architecture. Separate this engineering model from online, messaging and batch architectures. Make all models available throughout the design process so that developers question their users and determine which architecture is best for each business requirement.

The stream I-O engineering model will be blend many features of both online and batch engineering models, yet will integrate in the data access or domain model layer. This is particularly important since the business may well require immediate online access of the results of the stream IO processing.

Stream IO engineering model will support:

- Processing initiated by external messages or events, rather than user interaction or time.
- error processing which holds errors in a separate queue, rather than immediate notification of users,
- performance tuned for very high volume throughput with predictable access
- locking and data handling different from online use to allow concurrency with online

- centralized processing to maximize throughput
- constant execution of individual events.

Figure 4 illustrates the stream-IO engineering model.

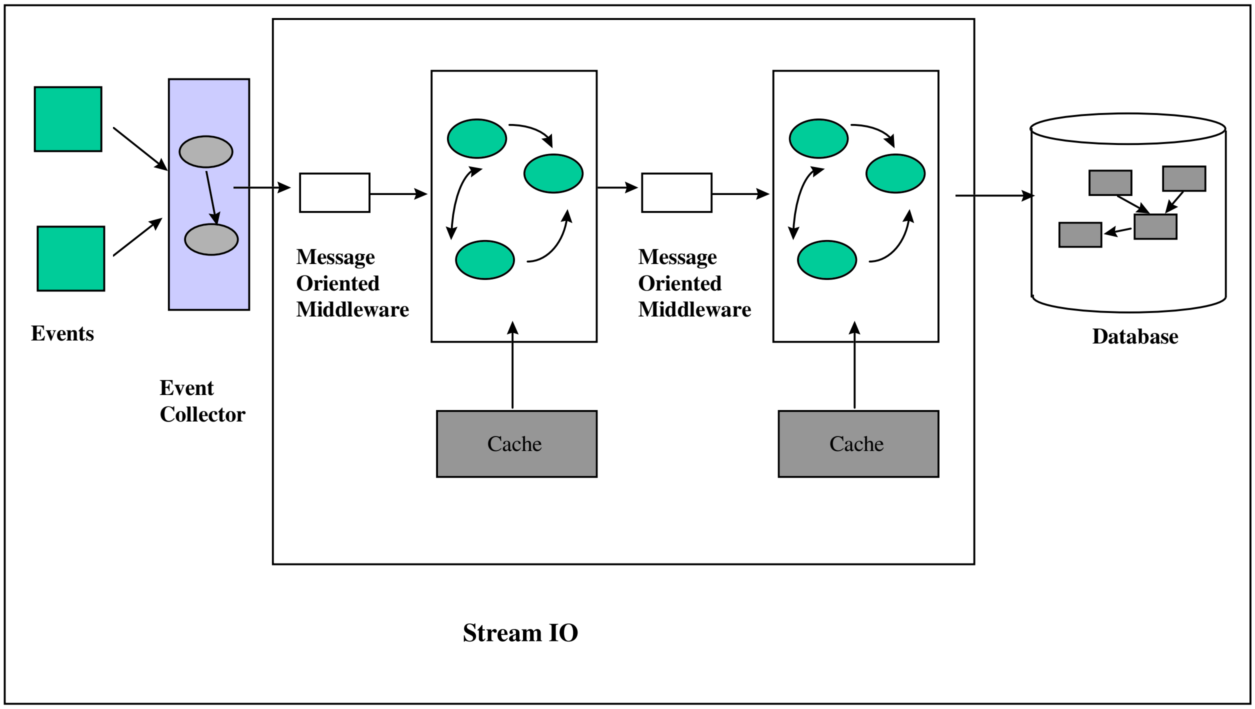

Figure 4

This figure illustrates the stream IO engineering model. Work is intiated by events, which may appear to happen constantly. The critical difference between this and batch is that work is initiated by events rather than time. Any distribution of processing would only be because of processing initiated by the event collector or middleware. This largely centralized processing provides very high throughput, as volumes for each individual batch function may be very high. These processes are predictable, and what is needed for each 'transaction' in the file of events can be predicted in advance. Again, the stream-IO engineering model uses the same underlying business object model and database as other models. The differences in the software architecture is in the distribution and initiation of system set-up classes. The Caches are used to retain as much information as possible in memory and thus speed up processing.

Example

In the telecommunications industry, call pricing is an example of a business function which this engineering model would support. An event happens – a person makes a call. That call can be polled immediately from the switch and priced. There is no business reason to wait and price the call later in a batch. In fact, the business needs to know the price of the call immediately in case the customer calls with a question.

Using the Stream IO engineering model, a rating engine continually receives call records from the switch, prices them and stores the priced call on the database for immediate access.

Known Uses and related patterns

This engineering model is technically the closest to the batch engineering model. The key differentiator is that these processes are initiated by an event, and batch processes are scheduled by time. This process is used in real time rating in the telecommunications industry.

Conclusion

This pattern language of four engineering models for software architecture defines architectures which together meet the requirements which characterize large scale business systems. These are summarized in the table below.

Characteristic	Requirements Satisfied by Each Engineering Model
User Interaction	High Medium None *Online Messaging Batch & Stream IO*
Predictability of Access	Little Medium Highly Predictable *Online Messaging Batch & Stream IO*
Object Model Structure	Complex Simple *Online & Messaging Batch & Stream IO*
Throughput	Low Low *Online & Messaging Batch & Stream IO*
User Response Time	High Medium-Some wait Medium-Query Answer Low *Online Messaging Stream IO Batch*
Process Initiation	User Interactive Event Event Time *Online & Messaging Stream IO Batch*
Distributed Processing	High Medium-Some Client Medium-Most Server Central *Online Messaging Stream IO Batch*

Acknowledgments

I am privileged to work with Chris Tatem and Mike Madsen, at AMS, both of whom conceived of and fought about these engineering models. They are the brilliant minds who are making these work. I also want to thank Liping Zhao, who shepharded this paper from a rough draft to a finished product.

References

1. F.Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern Oriented Software Architecture – A System of Patterns. John Wiley & Sons, 1996

2. M. Fowler. Analysis Patterns – Reusable Object Models. Addison Wesley, 1997

3. "Relational Database Access Layer", Proceedings PloP96.

4. Krasner, S.T.Pope. A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk 80. Journal of Object Oriented Programming 1(3), PP 26 - 49, August/September 1988, SIGS Purlications, New York, NY, 1988.