A Pattern Language for Workflow Systems

Authors:
Gerard Meszaros
Object Systems Group
87 Connaught Drive NW,
Calgary, Alberta, Canada T2K 1V9
e-mail: gerard.meszaros@acm.org
Phone: 1-403-210-2967

Kyle Brown
Knowledge Systems Corporation
4001 Weston Parkway, Cary NC 27513
e-mail: kbrown@ksccary.com

Abstract

This pattern language describes the process for creating any system which includes workflow as part of its requirements. It includes patterns for identifying the workflow requirements, for defining the architecture of the system, and for implementing that architecture.

1. Introduction

This pattern language describes the process for creating any system which includes workflow as part of its requirements. It includes patterns for identifying the workflow requirements, for defining the architecture of the system, and for implementing that architecture.

The patterns are organized into three main sections, Requirements, Architecture and Implementation. This is not to imply a strict timeline but to provide a partial ordering of the decisions to be made.

The language starts off with a pattern describing how to recognize that the system has Workflow Requirements. Next, the Architecture Patterns section provides a set of patterns for laying out the architecture of the workflow system, dividing it into WorkflowBusinessArchitecture specific patterns related to identifying the CentralWorkProduct and its states, and WorkflowTechnicalArchitecture patterns related to how the various subsystems are organized and communicate. Finally, the Workflow Implementation section describes patterns which can be used to implement the architecture in an extensible way.

2. Requirements Patterns

2.1.1 Workflow Requirements

Context:

You have been given a set of requirements for a system that involves shunting a document or other work-item between a number of users or related computer systems.

Problem:

What is the key aspect of these requirements?

Forces:

- Focusing on the wrong aspect of the requirements could lead you to an architecture which does not adequately handle the requirements.
- Requirements can be even harder to talk about crisply than design.
- Finding commonality between requirements of different systems requires excellent abstraction skills.

Solution:

When a system's main active responsibility is to act as a "chauffeur" or "guardian angel" for a particular document or other work-product through all or part of it's life-cycle, the system is said to be a "workflow system". A system which has such responsibilities in addition to others, it is said to have "workflow requirements". A workflow system often integrates a number of other (often "legacy") systems each of which has a specific subtask to accomplish. If a new system is organized as a series of subsystems each responsible for a particular task, the part of the system which routes the work around can be considered the "workflow manager".

What else can we say in defining "workflow"?

Record all the different way-points the work-item must be routed through. A commonly accepted way to record these way-points is with a data-flow diagram. Define the business processes (waypoints) that act on a document and then describe the flow of documents between the processes. Be sure to note exception cases where the documents can route back to previously encountered business processes.

Consider using a Workflow Architecture when implementing such a system.

Examples:

Many commercially available workflow systems (whether Object Oriented or otherwise) address this Requirements Pattern.

Microsoft Office components include the capability to route a document to a number of users via electronic mail. Thus, they implement “workflow requirements”.

Courier companies (such as Fedex) route a physical package through a number of physical subsystems (pickup and delivery vans receiving departments, warehouses, , inter-city trucks, customs brokers, airplanes). The waybill must reflect the changes in state of the actual package so that the user can locate it easily. The computer systems which support this business are likely arranged as “workflow system”.

3. Architecture Patterns

This section contains the patterns used to organize the architecture of the workflow system. It is divided into two key sections: Patterns related to characterizing the central workproduct and its states, and patterns related to defining the technical architecture of the workflow system.

3.1.1 Workflow Architecture

Context:

You have identified that your system has Workflow Requirements. You are defining the Application Architecture of your system.

Problem:

How should the application be structured?

Forces:

- An appropriate architecture makes solving the problem easy.
- Determining the best architecture may take a lot of work.
- Keeping track of where the document currently lives can be difficult.
- There are all sorts of “mechanics” issues related to moving the document around which are not directly related to the “architecture” of the workflow of the document.

Solution:

For a system with non-trivial workflow requirements, use a workflow architecture.

Divide the architecture into two topics areas.

The Workflow Business Architecture captures the states of the central workproduct and the valid transitions between them. It also assigns the subsystems responsible for the document in each state.

The Workflow Technical Architecture deals with the mechanics of delivering the document to a particular subsystem, keeping track of where it is, and keeping metrics on the number of documents being handled by each subsystem.

Examples:

Many commercially available workflow systems (whether Object Oriented or otherwise) implement this architectural Pattern.

Microsoft Office components include the capability to route a document to a number of users via electronic mail.

3.2 Workflow Business Architecture

This is an example of the pattern Business Architecture in [Meszaros97].

3.2.1 " Central WorkProduct(s)"

Context

One of the things that is sometimes lost in an object-oriented design is a sense of "location" of an object. In a workflow system the knowledge of what objects are where is absolutely crucial to the functioning of the system as a whole.

Problem

How do you pick out the "central" object that moves from person to person from the myriad of objects that are identified in an OOA&D effort?

Forces

- Managing System Complexity- it's easier to grasp the architecture of a system when you consider only a few of the most commonly encountered objects. People are easily lost in trying to understand all of the details of a number of objects.
- Sometimes, the central object is known by different names as the state changes. This makes it hard to identify as a single object.

Solution

There are two basic situations which you may encounter. These are:

1. A single workproduct which is routed to various subsystems, or
2. A composite workproduct consisting of several workproducts which are routed as a though they were a single object.

Single Workproduct

In some cases it will be obvious from the original analysis documents or user requirements documents which of the objects are your central "workproduct" objects. Look for active movement verbs like "route" or "send" to get an indication of which objects have a strong sense of location. Another key is to look for objects which appear to have different responsibilities over time – although normally in an OO-design process you would identify such objects as targets for splitting into multiple objects, what you may instead have is an object with a history. Such objects are wonderful targets for implementation with the State pattern (see Transform Process Steps into Workproduct States).

Composite Workproduct

If there are several potential objects with a strong sense of location, then you might in fact be dealing with a single "composite" workproduct that is a conglomeration of multiple workproducts. In this case, the composite acts a point of contact (or Facade) onto a subsystem of supporting objects, while the composite object manages the routing for the objects held within itself. Despite its name, this object is usually not a Composite as in [Gamma 95], since there is usually not a need for the recursiveness of the Composite pattern.

3.2.2 TransformedWorkProduct States

Context:

Workflow requirements are usually phrased in terms of the processes that are involved. In many cases they are drawn using standard dataflow diagrams showing data (the WorkProducts involved) moving between processes.

Problem:

How do you develop an object-oriented representation of the processing steps of a document?

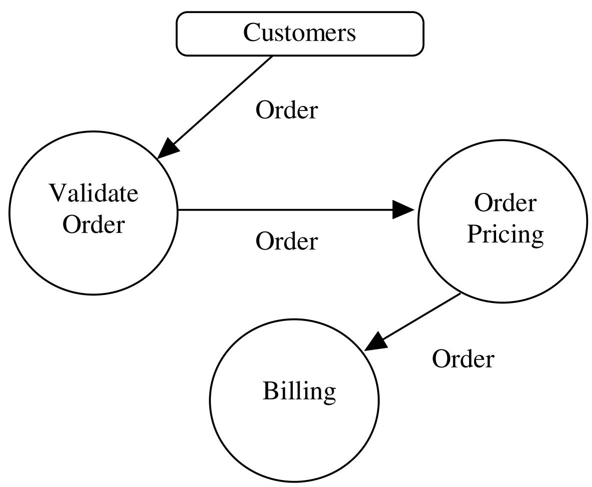
Forces:

- Maintaining conceptual integrity of the analysis- Information gained in the analysis stage should not be lost during design.

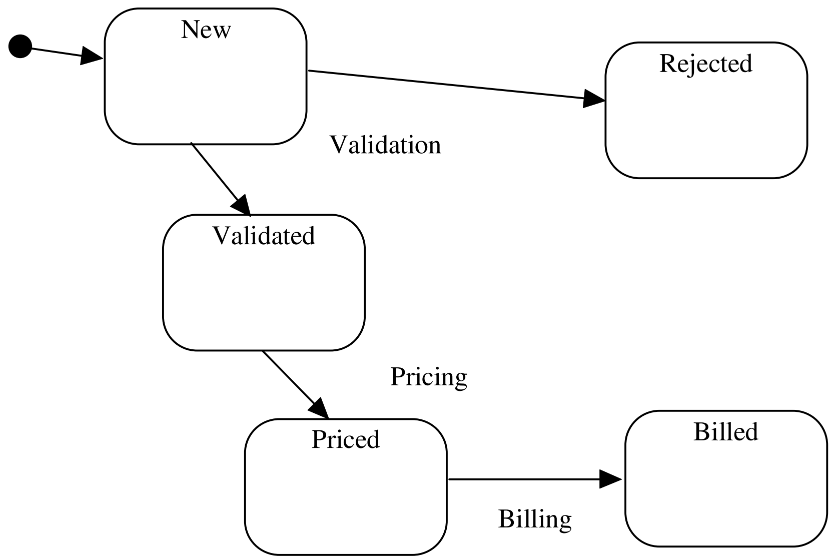
• Arriving at a maintainable design representation of the analysis artifacts- The overall structure of the final design should be easily understood and extended.

Solution:

Begin by taking the dataflow representation of the process, or by drawing one if one has not yet been prepared. In a standard dataflow diagram [Yourdon 89], the processes are represented by circles (nodes) while the data flowing between the processes are represented by arrows (arcs). An example is shown below:



To derive the state of the WorkProduct from the dataflow diagram we must consider these nodes and arcs to be the states in a state transition diagram. The key transformation step is to transform the arcs in the previous diagram to be nodes and the nodes to be arcs. In essence what is involved in transforming the state machine from a Mealy model to a Moore model. In our example we will use the notation from [Booch 94] Consider the following transformation of the previous diagram:



This transformation results in a description of the states of the Workflow Product, which can then be implemented using a table-driven approach, or the State Pattern from [Gamma 95]. These states are crucial to understanding the response of the WorkProduct objects to the different messages that may be sent to them through their lifecycle.

The advantage is that is easy to maintain either approach – updating a table is an easy enough task, while even the class-driven approach of the state pattern is simple to maintain since the state-dependent behavior of the WorkProduct object is localized to a single hierarchy of classes. Importantly, this approach also maintains the conceptual integrity of the analysis model – the processes involved still remain as part of the behavior of the WorkProduct.

Examples:

We have applied this pattern in many systems in our consulting practices. [Brown96] is a documented application of the pattern in an order processing system.

3.3 Workflow Technical Architecture

This section introduces the various approaches to building a workflow system and describes in more detail the technical architecture patterns for building a Push Workflow system. It is an example of the pattern Technical Architecture in [Meszaros97].

3.3.1 Push Workflow Architecture

Context

You have determined the states of the central workproduct and the subsystems to which the workproduct should be routed. You would like to make the definition of the workflow easy to understand and manage.

Problem

How do you organize the components of the system? How do the components interact and hand off the workproduct

Forces

- Having all subsystems know about the other subsystems introduces excessive coupling into the system.
- Centralizing the knowledge of the workproduct routing results in a single object which must be aware of all other objects. This makes it easy to manage the workproduct state space but introduces some coupling within the system.
- A distributed model may be more flexibile but it can be hard to ensure that the workproduct state model is "live" (does not stall anywhere.)

Solution

Use a single central workproduct router which pushes the workproduct to each of the processing systems at the appropriate point in the lifecycle of the workproduct.

Resulting Context

By choosing the push model, you have centralized the management of the workproduct state. This could easily become a bottleneck in that the workproducct router needs to be aware of all the subsystems, You can ensure that this does not create unnecessary coupling by using a Standard Subsystem Interface to minimize interface coupling with the subsystems, and by using the State Object pattern to make the definintion of the workproduct state machine easy to change.

3.3.2 Pull Workflow Architecture

Context

You have determined the states of the central workproduct and the subsystems to which the workproduct should be routed. You would like to make the definition of the workflow highly flexibile and dynamically modifiable as workproduct processing subsystems are added or removed.

Problem

How do you organize the components of the system? How do the components interact and hand off the workproduct

Forces

- Having all subsystems know about the other subsystems introduces excessive coupling into the system.
- Centralizing the knowledge of the workproduct routing results in a single object which must be aware of all other objects. This makes it easy to manage the workproduct state space but introduces some coupling within the system.
- A distributed model may be more flexibile but it can be hard to ensure that the workproduct state model is "live" (does not stall anywhere.)

Solution

Use a single central workproduct repository which holds the workproduct until requested by a processing subsystem. When a processing subsystem joins the workflow system, it expresses interest in specific events of the relevant workproduct(s) by registering with the workproduct repository. When a workproduct is changed by one processing subsystem in a way which causes an event to be signalled, the registered subsystems are informed (in sequence, using Chain Of Responsibility)

Resulting Context

By choosing the pull model, you have completely de-centralized the management of the workproduct state. This can make it hard to ensure that a workproduct will not stall in some non-terminal state because no processing subsystems are interested in the most recently signaled event. It may be necessary to run a test workproduct through the system after every change in configuration. It is very flexible but it can be hard to ensure that the workproduct state model is "live" (does not stall anywhere.)

Solution

Use a single central workproduct router which pushes the workproduct to each of the processing systemsat the appropriate point in the lifecycle of the workproduct.

Resulting Context

By choosing the push model, you have centralized the management of the workproduct state. This could easily become a bottleneck in that the workproduct router needs to be aware of all the subsystems, You can ensure that this does not create unnecessary coupling by using a Standard Subsystem Interface to minimize interface coupling with the subsystems, and by using the State Object pattern to make the definition of the workproduct state machine easy to change.

3.3.3 Standard Subsystem Interface

Context

You have determined the states of the central workproduct and the subsystems to which the workproduct should be routed. You have chosen to use a “push” workflow architecture for you system to deliver the workproduct to the various workproduct processing subsystems

Problem

How do you actually deliver a workproduct to a subsystem?

Forces

- Having all subsystems know about the other subsystems introduces excessive coupling into the system.
- A standard interface may introduce additional development effort when federating pre-existing systems.

Solution

Provide a standard interface for all subsystems. The interface should provide operations for delivering and receiving workproducts from the subsystems. If metrics reporting is required, this should be included in the interface. Inquires on particular workproducts should also be supported.

Ensure that all subsystems implement this interface.

3.3.4 Proxy per Remote Subsystem

Context

You have defined a Standard Subsystem Interface

Problem

How do you deliver a workproduct to an subsystem which exists in another computer?

Forces

- Neither the workproduct nor the other subsystems should be aware of which parts of the process are implemented in the same computer.
- A remote subsystem may not always be available and handling this should not be a responsibility of the object interacting with the subsystem (too much coupling.)

Solution

Create a Remote Proxy Object for each non-local subsystem. This could be any one of Remote Proxy, Caching Proxy or Half-Object Plus Protocol (HOPP), depending on the performance and availability requirements of the Proxy. (E.g. If the proxy needs to be able to respond to queries even when the remote subsystem is unavailable, use HOPP and keep the answers to all possible queries synchronized with the remote subsystem.)

The actual delivery of information and synchronization requests to the remote system can be implemented using an Object Request Broker which implements the Remote Proxy pattern (such as CORBA) or via Remote Procedure Calls (RPCs). In high availability systems, a "persistent messaging" system such as IBM's MQSeries or DEC's DecMessaging can be used to ensure that the work-product is not lost in transit due to a server (computer) outage.

4. Workflow Implementation Patterns

This section describes how to implement the workflow architecture described in the previous section. Once again, the patterns are divided into Business Logic and Technical Infrastructure.

4.1 Business Logic Implementation

4.1.1 " WorkProduct State Objects"

Context

You have described a workproduct's "waypoints"as states using the "Transformed Workproduct States"pattern.

Problem

How do you represent these states in an OO language in an easily extensible and maintainable way?

Forces

- Code should reflect the state information shown in the design and analysis for clarity.
- Code which is all in one place is easier to understand.
- Code which is factored into smaller pieces is easier to modify and especially to extend.

Solution

Use the "State Pattern"[Gamma 95] to take the previously defined state machine and turn it into classes. Two benefits are drawn from this:

The workproduct now has a single, unique state that can be tested directly. This eliminates a great deal of "conditional"code that would otherwise be coded into the workproduct.

Many of the state-dependent behaviors of the workproduct may be deferred to the state classes themselves. This will reduce the size and complexity of the workproduct implementation.

[Ryan 97] and [Brown 95] describe uses of this pattern for implementing state-based behavior.

Resulting Context

The implementation of the workproduct using the state pattern can be easily extended by adding more state classes or by changing the interface of the state classes to add additional behavior. In this way the external interface of the workproduct can remain stable while new features are added.

4.2 Technical Infrastructure Implementation

4.2.1 Inbox/Outbox per subsystem

Context

You are implementing a Push Workflow Architecture. You have defined a standard interface to which all subsystems will comply.

Problem

How do you implement the workproduct delivery mechanism?

Forces

- A subsystem may not always be ready to accept the workproduct.
- A subsystem should be able to relinquish ownership of the workproduct as soon as it is completed processing.
- To facilitate tracking of workproducts and gathering of statistics, it should be possible to keep track of where the workproduct was last sent.

Solution

As part of the standard interface for each subsystem, create an inbox and an outbox. These FIFO queues provide operations to add and remove workproducts, and keep track of how many workproducts are in them. When the decision is made to deliver the workproduct to a particular subsystem for the next step of processing, place the workproduct into the subsystem's inbox (via an operation provided in the Standard Subsystem Interface.

The subsystem gets the first workproduct in its inbox and begins processing. When the subsystem is finished processing the workproduct, it updates the state and puts the workproduct into it's outbox. The workproduct state machine determines what subsystem it should be routed to next and places it into that subsystems inbox. This process continues until the workproduct is placed into the inbox of the special Processing Terminated Subsystem. (This should probably be a pattern. There is probably another special subsystem called Entry Point Subsystem which accepts work from outside the workflow system and presents it to the Workflow Router by putting it into it own outbox.)

Resulting Context

We now have a system where all communication between subsystems is via queues. To ensure that no workproducts are lost during system outages, it may be necessary to make the queues persistent and transactional. Transactional Queue ensures that a workproduct is not actually removed from it until the transaction that changes the workproduct state is successfully committed. Persistent Queue ensures that the contents of a queue are not lost if the processor which holds the queue suffers a failure. MOM systems such as MQSeries and DecMessaging implement the Persistent Queue pattern.

5. References

[Booch 94] Grady Booch, *Object-Oriented Analysis and Design With Applications*, Addison-Wesley, Reading, MA, 1994

[Brown 96] Kyle Brown, "Experiencing Patterns at the Design Level", *Object Magazine*, January 1996, 5(9) p. 44-52

[Gamma 95] Erich Gamma, Richard Helms, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA, 1995

[Meszaros97] Gerard Meszaros, *"Archi-Patterns: A Process Pattern Language for Defining Architectures"*, proceedings of PLoP97

[Ryan 97] Patrick Ryan, "Handling Exceptional Behavior with State Objects", *The Smalltalk Report*, June 1997, 6 (8), pp. 16-19

[Yourdon 89] Edward Yourdon, *Modern Structured Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1989