A Large Scale Aspect System

Therapon Skotiniotis

Department of Computer Science
Northeastern University
Boston, Massachusetts 02115, USA
skotthe@ccs.neu.edu

1 Introduction

Things that need to be done :

- Understand and explain how the system by ABB works in a more clear and less detailed manner than what is provided by their documentation.
- Generate a mapping for ABB between their terminology and the general AOOD research terminology.
- Identify areas where new technology or existing AOP ideas will benefit ABB's system more.
- The converse to the above point might be true. If so then I have to identify and explain the reasons why.
- Issues that the ABB system attacks and reasons why they are considered to be "aspectual".
- Introduce new issues and propose solutions to them. Proposed solutions should provide better Separation of Concerns, as well as, not to compromise the extensibility/adaptability/reusability of existing and new features.

2 The System

What follows is a general overview of what the system is, as far as I understand it

The Aspect Object System (as coined by the ABB documents) targets to fullfil the following goals:

- A seamless, to the user, interface that enables views of different components of the system through a point and click interface. Navigation through the components as well as the information that these components manipulate are encapsulated as "aspects".
- New "aspects" can be loaded on demand (from the existing aspect repository or through the installation of new aspects into the system).
- Changes made by the user on specific "aspects" are reflected to all other instances of that aspect within the system automatically.
- Fault tolerance, security and scalability is ensured through well defined interfaces defined by the system (Afw) and enforced by the implementation of all "aspects".
- A hierarchical approach in grouping "aspects", components and their relations allows for easy management, deployment and development through out the system.
- More to come here !

This is a sketch of some of the important operations that the system goes through. Most of what follows in this section (for now) is from examples in the documents along with some software engineering terminology being mapped to these operations.

"Aspects" within the ABB frame of mind are encapsulation of operations that hold, the data members needed for this operations as well as the functions / procedures that are available for their manipulation. An "Object", again in the ABB frame of mind, is a component acting as a "holder" for the different aspects that affect this entity. That is, an object is a system entity (a pump, motor etc.) that holds all necessary aspects that deal with its operation. This objects are arranged in a hierarchical order. Motors, for example, are grouped together in the "Motor Group". This grouping itself is handled as an aspect inherited to all the members of the motor group.

"Aspects" within the system have their own structure and categorization within the "Aspect System". The "Aspect System" holds information about the relationships between aspects, a hierarchical approach is also maintained within the "Aspect System" that groups aspects together, also "types" are provided for aspects as well as information regarding inheritance and interfaces that are implemented by aspects.

Sketching the process of adding or activating an aspect within the system goes through the following procedure. Aspects, when they are first introduced into the system, go through a register and publish process with the Aspect System. That is an aspect registers itself with the Aspect System providing its "type", its inheritance specification (if it inherited from some other aspect type, or if itself can be used to inherit other aspects), and interfaces that the aspect is making available to the whole system.

The next step is the definition of "Objects" that will in essence hold aspects which identify the "Object's" operations and services. Once an aspect is activated (i.e. the user clicked on an object's aspect in order to view or alter information) the "Aspect System" will then provide a .COM object representing this aspect with all its implementable interfaces. The above statement hides a lot more information about security, authorization "Service Managers" and others in order to provide a simple first attempt in explaining the systems behavior when it comes to aspects. At a later stage these hidden information will be addressed.

For each request to an aspect the system created a request manager. The request manager creates .COM objects for in order to service the requests being accepted from its client. The service manager acts as a communication layer between the client and the Aspect System. A request from the the client is propagated to its request manager, who, in turn, requests from the Aspect System a pointer to the aspect type that needs to be instantiated in order to service the clients request. A .COM object is then created holding the definition of the aspect that the Aspect System returned. This .COM object (Aspect Object) will delegate requests according to the aspects definition (interfaces) and their bindings (other dependent functions that need to be called) to satisfy a specific request.

Adding or removing aspects from the system is thus a matter of removing or creating the corresponding .COM object within the systems state at the time. Features tendered by the .COM technology are also exploited within ABB's system so as to allow distribution of the .COM object along with its state to other "System Objects". This allows for object interaction based on the same aspect currying the aspects state.

Although this seems to be highly complex to maintain within the system (due to COM object being moved from point to point within a possibly large system) the well designed "Aspect System" takes care of this issue. The fact that all aspects have a type along with a unique identifiers (Group as well as individual) allows for easy tracking albeit at the cost of maintaining this information. Another issue to note here is that since all objects whithin the system are maintained in a hierarchical manner, objects also have groups as well as their own identifiers. As such the system allows a mapping from "Objects" to "Aspect Instances". Invocation is then a matter of a query to the table of identifiers of each object to find the corresponding aspect object, and then interface, to be executed.

The above is a simple description of the aspects interaction with objects. Keeping in mind that the whole process is taking place on top of a networked system with a given topology. Interactions also have to go through a "System Level" that encapsulates topology transparency, security, fault tolerance etc. However since these issues are present through out any such system, they are excluded (for now !!).

3 Terminology Mapping

Map the terminology ... I'll try to start this first .. this will give immediate results as well as help me map certain terminology from the start so that I can understand the system with more ease (or at least I hope it will).

ABB Terms	*AOP / S.E. Terms*	*Notes*
Object	Adapter	No expected fields/methods
Aspect	Aspect	
Aspect Framework	Base System	
Aspect Object	Meta-Object for Aspect	Proxy /Delegates
Aspect System	Aspect Repository	Appears to be itself aspectual
Aspect Type	Base Aspect	
Object Manager	Factory/Proxy/Delegates	
Structure Hooks	???	
Aspect Group	Specification of a Collaboration	Only for Aspects
Object Type Groups	Collaboration	Only for Object
Object Type Rules	Pointcut Designators	
Structures	Aspects	
Integration Level	Pointcut designator	
Aspect Category	Aspect Grouping	Maps to aspect type
Object Types	Abstract Object Classes	

Not all terms are mapped yet. Also some of the terms might change over time as I get more familiar with the system itself. Certain choices had to be made in order to keep the table simple but also give more concrete mappings between the two Domains of terminology.

3.1 Discussion on Terminology

Object – Adapter Aspect Container could be a more relaxed explanation of Object here, although Adapter brings it closer to the ideas of Aspectual Collaborations since similarities between the 2 ideas are evident.

Aspect – Aspect Using Aspect fits with the conceptual usage of the term although the actual implementation of this term in the system might be a typical OO object.

Aspect Framework – Base System Aspects in the AOP community are defined as the self contained modular units of crosscutting implementation. In order to have a crosscutting implementation this implies that a base level implementation, providing the basic minimum functionality of a system, exists. Based on this base level implementation certain concerns that need to be addressed are not well localized. This non-localized concerns, when abstracted into their own entities become your aspects. With ABB's system the Aspect Framework (Afw) provides the basic features that any version of the system must have(concepts, API, tools, etc.). The different applications for each hardware component, for example pump alarms, motor temperature and speeds, are viewed as the aspects that need to interact on top of the Afw but also amongst themselves.

Aspect Object – Meta-Object This is a bit more complex and certain parts of it are a bit unclear for me. However it appears to be the main delegating object that services the requests made by the users application to the corresponding aspect object. The Aspect Object appears to intercept calls to aspects and re-route them according to the interface and aspect interaction specification given to it.

Aspect System – Aspect Repository A repository that holds the aspects definitions for the system in question. Sub-categorization within the Aspect System enables grouping as well as sharing of aspects between systems.

Aspect Type – Base Aspect Due to the repository layout enriched with groupings and querying facilities, there seems to be a layered approach on how to identify the correct aspect for the correct call. The aspect type structure maps an aspect directly to the request that needs to be serviced.

Aspect Group The system allows you to define Aspect Groups that hold group of aspects and their interactions between them. More like a collaboration between aspects viewed as a component.

Object Types Objects that are of the same type (motors, pumps tanks) can be grouped to form an Object Type. Rules within types give a global restrictions as well as interactions that all members of the group have to implement. A collaboration of objects along with their interactions that abstracts common issues and allows for easier reuse.

4 Questions / Problems

- It appears that in order to actually execute a method/function that is part of an aspect we have to propagate through the Aspectual System. The order of propagation is unclear to me, from the documents.
- The Aspect Object appears to be defined in a lot of places within the document, in each place different functionality is added to it. I am not clear as to how this objects works and what its complete role is in the System ? Would it be possible to have an example ?
- From the examples found within the document, I could not see how the system handles aspects that are co-dependent. That is non-orthogonal. For instance scheduling and synchronization within a system are considered to be non-orthogonal since first we need to synchronize a request to a shared resource and then based on the result of the synchronization aspect call the scheduling aspect to schedule the request. How would a similar scenario (it does not have to be synchronization and scheduling) be addressed with your system ?
- Would it be possible to have a code example of a simple system that contains several aspects and objects (3 - 4) distributed over a couple of nodes (computers).
- It appears that there is a lot of querying within the system in order to get object/aspect id's as well as interfaces that are implemented by components. How does the system deal with lost messages between

components ? Does it deal with this within the base level framework or through aspects ?

- Can you add new aspects/functionality to a system at runtime ? I could not get an answer to this from the documentation that I had available. If I can add aspects at runtime, how does the system handle replacement of an active aspect by a newly inserted replacement aspect ?

- Where does the programmer specify the order of events on a specific operation. For example, if I have 2 users on my system at 2 different terminals with different authorities and a pump in my factory signals an alarm. Assume that both attempt to first view the alarm first, then the status of the pump. After that they both decide to alter the configuration of the pump in order to fix the problem that caused the alarm. Where would I place my code for synchronizing the operations? Is it within embedded within the method that is responsible for changing the configuration of the pump, or is it in a separate aspect all together ?

- As a test example consider the following situation. A specific device has an On/Off button. Encapsulating the operations of turning the device On or Off as an aspect could you extend the application with extra aspects so that you can check whether the On button was pressed twice and guard against it. Could we introduce a logging aspect that logs the last operation on the device. Then the On operation will check the logging aspect to see what type of operation was executed last. If the last operated action on the device was an "On" action then issue an error, else turn the device on. A simple example, although if you would like to add more sense to the example consider a double click operation on a device as being a different function operation as a single click. How would one go about to implement this example given first the implementation of the single click function and then extending it to handle the double click functionality in the ABB system.