Information extracted from Implicit Invocation Architecture Pattern

We are providing this information to facilitate the architecture evaluation. We encourage you to use the information provided here during the ATAM exercise.

<table>
<tr><td colspan="3">Pattern Name: Implicit Invocation</td><td>Pattern Type: Architecture</td></tr>
<tr><td colspan="2">Brief description</td><td colspan="2">Implicit invocation is structured around event handling, using a form of callback. The idea is that instead of invoking a procedure directly, a component can announce one or more events. Other components in the system can register an interest in an event by associating a procedure with the event. When the event is announced the system itself invokes all of the procedures that have been registered for the event. Thus an event announcement implicitly causes the invocation of procedures in other modules.</td></tr>
<tr><td colspan="2">Context</td><td colspan="2">An object which delegates work to another object by using its services becomes dependent on it. Therefore, it must ensure its own consistency with respect to the objects on which it depends.</td></tr>
<tr><td colspan="2">Problem description</td><td colspan="2">Changes of one object often require dependent objects to change accordingly. Making every object explicitly inform every dependent object about its state changes intertwines object interfaces and implementations, thereby hampering system evolution and maintenance.</td></tr>
<tr><td colspan="2">Suggested solution</td><td colspan="2">Components (or objects) do not know each other explicitly. Rather, they communicate by announcing events. Objects register for events, so that announcing an event by one object leads to the notification of those objects which have registered for the event.</td></tr>
<tr><td colspan="2">Forces</td><td colspan="2">If the changed object were to inform its dependent objects by explicit operation invocations, the changed object's interface and implementation would become dependent on its depending objects as well. This introduces cyclic dependencies and should be avoided since it hampers system evolution and maintenance</td></tr>
<tr><td colspan="2">Available tactics</td><td colspan="2">Decouple dependent objects from those objects which they depend on</td></tr>
<tr><td colspan="2" rowspan="2">Affected Attributes</td><td>Positively</td><td>Negatively</td></tr>
<tr><td>Maintainability: add or replace components with minimum affect on other components.
Reusability</td><td>Complexity: Difficult to control the processing order of the implicit invoked modules</td></tr>
<tr><td rowspan="3">Supported general scenarios</td><td>S1</td><td colspan="2">When replacing an object that depends on a number of other objects, the replacement will not affect the objects that it depends on.</td></tr>
<tr><td>S2</td><td colspan="2">When add a new object that needs to get informed about specific state changes of other objects, the system should make it easy to add such a object.</td></tr>
<tr><td>S3</td><td colspan="2">When the notification process due to an object’s change of state has to be carried out, it can be done anonymously and observers and subjects are statically decoupled.</td></tr>
<tr><td>Examples of usage</td><td colspan="3">GUI “Callback” subroutines written to handle mouse movement, clicks on buttons, and so on.</td></tr>
</table>

Information extracted from Layered Architectural Pattern

We are providing this information to facilitate the architecture evaluation. We encourage you to use the information provided here during the ATAM exercise.

<table>
<tr><td colspan="3">Pattern Name: Layered</td><td>Pattern Type: Architecture</td></tr>
<tr><td colspan="2">Brief description</td><td colspan="2">The Layers pattern helps to structure applications that can be decomposed into groups of subtasks in which each group of subtasks is at a particular level of abstraction.</td></tr>
<tr><td colspan="2">Context</td><td colspan="2">You are designing a complex enterprise application that is composed of a large number of components across multiple levels of abstraction.</td></tr>
<tr><td colspan="2">Problem description</td><td colspan="2">How do you structure an application to support such operational requirements as maintainability, reusability, scalability, robustness, and security?</td></tr>
<tr><td colspan="2">Suggested solution</td><td colspan="2">Separate the components of your solution into layers. The components in each layer should be cohesive and at roughly the same level of abstraction. Each layer should be loosely coupled to the layers underneath.</td></tr>
<tr><td colspan="2">Forces</td><td colspan="2">• Localizing changes to one part of the solution minimizes the impact on other parts
• Components should be reusable by multiple applications.
• Individual components should be cohesive
• Unrelated components should be loosely coupled.</td></tr>
<tr><td colspan="2">Available tactics</td><td colspan="2">Maintain semantic coherence
Generalize the module
Information hiding</td></tr>
<tr><td colspan="2" rowspan="2">Affected Attributes</td><td>Positively</td><td>Negatively</td></tr>
<tr><td>Flexible, modifiability, reusability, Testability</td><td>Performance
Increased complexity for simple application</td></tr>
<tr><td rowspan="3">Supported general scenarios</td><td>S1</td><td colspan="2">The client application wants to consume a set of services offered by the server. The services of the server are exposed by the topmost layer. The client will have no direct knowledge of any lower layers.</td></tr>
<tr><td>S2</td><td colspan="2">The lower layer monitors the state of a system. When it detects a change, it fires an event exposed by a component from the upper layers.</td></tr>
<tr><td>S3</td><td colspan="2">The changes in one layer only affect the layers above and below it.</td></tr>
<tr><td>Examples of usage</td><td colspan="3">OSI network (seven layers) architecture
Virtual machine</td></tr>
</table>

Information extracted from Model View Controller (MVC)

We are providing this information to facilitate the architecture evaluation. We encourage you to use the information provided here during the ATAM exercise.

<table>
<tr><th colspan="2">Pattern Name: MVC</th><th colspan="2">Pattern Type: Architecture</th></tr>
<tr><td>Brief description</td><td colspan="3">MVC pattern isolates business logic from user interface considerations, resulting in an application where it is easier to modify either the visual appearance of the application or the underlying business rules without affecting the other.</td></tr>
<tr><td>Context</td><td colspan="3">The purpose of many computer systems is to retrieve data from a data store and display it for the user. After the user changes the data, the system stores the updates in the data store. Because the key flow of information is between the data store and the user interface, you might be inclined to tie these two pieces together to reduce the amount of coding and to improve application performance.</td></tr>
<tr><td>Problem description</td><td colspan="3">However, this seemingly natural approach has some significant problems. One problem is that the user interface tends to change much more frequently than the data storage system. Another problem with coupling the data and user interface pieces is that business applications tend to incorporate business logic that goes far beyond data transmission.</td></tr>
<tr><td>Suggested solution</td><td colspan="3">The Model-View-Controller (MVC) pattern separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes:
• Model. The model manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).
• View. The view manages the display of information.
• Controller. The controller interprets the mouse and keyboard inputs from the user, informing the model and/or the view to change as appropriate.</td></tr>
<tr><td>Forces</td><td colspan="3">Support for different ‘look & feel’ standards required.
Runtime user interface changes required.
Supporting multiple views.</td></tr>
<tr><td>Available tactics</td><td colspan="3">Localize modification
Use an intermediary to prevent ripple effects</td></tr>
<tr><td rowspan="2">Affected Attributes</td><td colspan="2">Positively</td><td>Negatively</td></tr>
<tr><td colspan="2">Flexible, modifiability</td><td>Increased complexity, inhibit reuse</td></tr>
<tr><td rowspan="3">Supported general scenarios</td><td>S1</td><td colspan="2">When the system needs to present the same data using different views, the data should be presented consistently.</td></tr>
<tr><td>S2</td><td colspan="2">When the system is going to present the data to a new device, the system should be can easily modified to satisfy such change.</td></tr>
<tr><td>S3</td><td colspan="2">When system is required to support several user-interface paradigms, it can be easily implemented.</td></tr>
<tr><td>Examples of usage</td><td colspan="3">Web application
Struts framework</td></tr>
</table>

Information extracted from Pipe and Filter Architecture Pattern

We are providing this information to facilitate the architecture evaluation. We encourage you to use the information provided here during the ATAM exercise.

<table>
<tr><th colspan="3">Pattern Name: Pipe and Filter</th><th>Pattern Type: Architecture</th></tr>
<tr><td colspan="2">Brief description</td><td colspan="2">The architecture consists of a chain of processing elements (processes, threads, and so on), arranged so that the output of each element is the input of the next.</td></tr>
<tr><td colspan="2">Context</td><td colspan="2">You have an integration solution that consists of several financial applications. The applications use a wide range of formats—such as the Interactive Financial Exchange (IFX) format, the Open Financial Exchange (OFX) format, and the Electronic Data Interchange (EDI) format —for the messages that correspond to payment, withdrawal, deposit, and funds transfer transactions. Integrating these applications requires processing the messages in different ways.</td></tr>
<tr><td colspan="2">Problem description</td><td colspan="2">How do you implement a sequence of transformations so that you can combine and reuse them independently?</td></tr>
<tr><td colspan="2">Suggested solution</td><td colspan="2">Implement the transformations by using a sequence of filter components, where each filter component receives an input message, applies a simple transformation, and sends the transformed message to the next component. Conduct the messages through pipes that connect filter outputs and inputs and that buffer the communication between the filters.</td></tr>
<tr><td colspan="2">Forces</td><td colspan="2">Many applications process large volumes of similar data elements.
The processing of data elements can be broken down into a sequence of individual transformations.</td></tr>
<tr><td colspan="2">Available tactics</td><td colspan="2">Prevent ripple effects by maintaining existing interface
Restrict communication path</td></tr>
<tr><td colspan="2" rowspan="2">Affected Attributes</td><td>Positively</td><td>Negatively</td></tr>
<tr><td>Reusability
Modifiability
Improved performance (Typically, filters do not wait for a scheduling component to start processing)</td><td>Increased complexity of assessing the state
Increased complexity due to error handling.
Lowered performance (Communication overhead. Transferring messages between filters incurs communication overhead)</td></tr>
<tr><td rowspan="3">Supported general scenarios</td><td>S1</td><td colspan="2">The system needs to covert the data from one format to another format. The conversation needs several intermediate stages/processes.</td></tr>
<tr><td>S2</td><td colspan="2">The implementation of each transformation can be reused</td></tr>
<tr><td>S3</td><td colspan="2">The configuration of the transformations can be changed dynamically</td></tr>
<tr><td>Examples of usage</td><td colspan="3">Applications to convert files in PS format to PDF format</td></tr>
</table>

Information extracted from Task Control Architecture Pattern

We are providing this information to facilitate the architecture evaluation. We encourage you to use the information provided here during the ATAM exercise.

<table>
<tr><td colspan="3">Pattern Name: Task Control</td><td>Pattern Type: Architecture</td></tr>
<tr><td colspan="2">Brief description</td><td colspan="2">Task Control Architecture a "hybrid" architecture, which is composed of robot-specific modules which communicate through a general purpose central control module which is common in all systems. This central control module includes a task tree, where task sequences and concurrency is planned, and modules are activated/deactivated according to various constraints. TCA has a major emphasis on constraints on coordination of control of planning, perception, action, and monitoring processes. TCA is viewed as a high level <u>operating system</u> that provides an integrated set of commonly needed control functions to support distributed communication, task decomposition, resource management, execution monitoring, and error recovery.
TAC consists of distributed modules that communicate via message passing through a central server (a.k.a. central control module). It uses Implicit Invocation of modules:
Modules do not communicate directly
Central server multicasts messages to modules that register for them
Modules may also use execution monitors that register condition-action pairs with server
As the control functions provided by TCA are only available in C or LISP, we are not going to use TCA as a framework/platform for our Robot system. We only use the Implicit Invocation idea there. So, please refer to Implicit Invocation Architecture Pattern for later sections.</td></tr>
<tr><td colspan="2">Context</td><td colspan="2">Refer to Implicit Invocation</td></tr>
<tr><td colspan="2">Problem description</td><td colspan="2">Refer to Implicit Invocation</td></tr>
<tr><td colspan="2">Suggested solution</td><td colspan="2">Refer to Implicit Invocation</td></tr>
<tr><td colspan="2">Forces</td><td colspan="2">Refer to Implicit Invocation</td></tr>
<tr><td colspan="2">Available tactics</td><td colspan="2">Refer to Implicit Invocation</td></tr>
<tr><td colspan="2" rowspan="2">Affected Attributes</td><td>Positively</td><td>Negatively</td></tr>
<tr><td>Refer to Implicit Invocation</td><td>Refer to Implicit Invocation</td></tr>
<tr><td rowspan="3">Supported general scenarios</td><td>S1</td><td colspan="2">Refer to Implicit Invocation</td></tr>
<tr><td>S2</td><td colspan="2"></td></tr>
<tr><td>S3</td><td colspan="2"></td></tr>
<tr><td>Examples of usage</td><td colspan="3">Refer to Implicit Invocation</td></tr>
</table>