1.1 Asynchronous Business Process Pattern

1.1.1 Name and Source

Asynchronous Business Process pattern
Page 153-167 in the book "Web Service Patterns: Java Edition" [WSP 03]

1.1.2 Also Known As

-

1.1.3 Type

Micro-architectural design pattern

1.1.4 Intent

To let the client execute a long-running Web Service and return quickly without locking the client process.

1.1.5 Problem

Human users of an application GUI, e.g. a web browser, do seldom have more patience than to wait for 10 seconds until they get some kind of response. However, the entire business process does not necessarily have to be finished within such a short period of time but often it is acceptable to get a notification later about the result e.g. an email message. It is also a desirable feature to be able to check the current state of a long-running business process that may involve many different web services from different companies as one aggregated business process exposed as a Web Service.

How can you provide a long-running business process as a web service that returns quickly without blocking the client, while also letting the client be able to check the current state of the business process ?

1.1.6 Forces

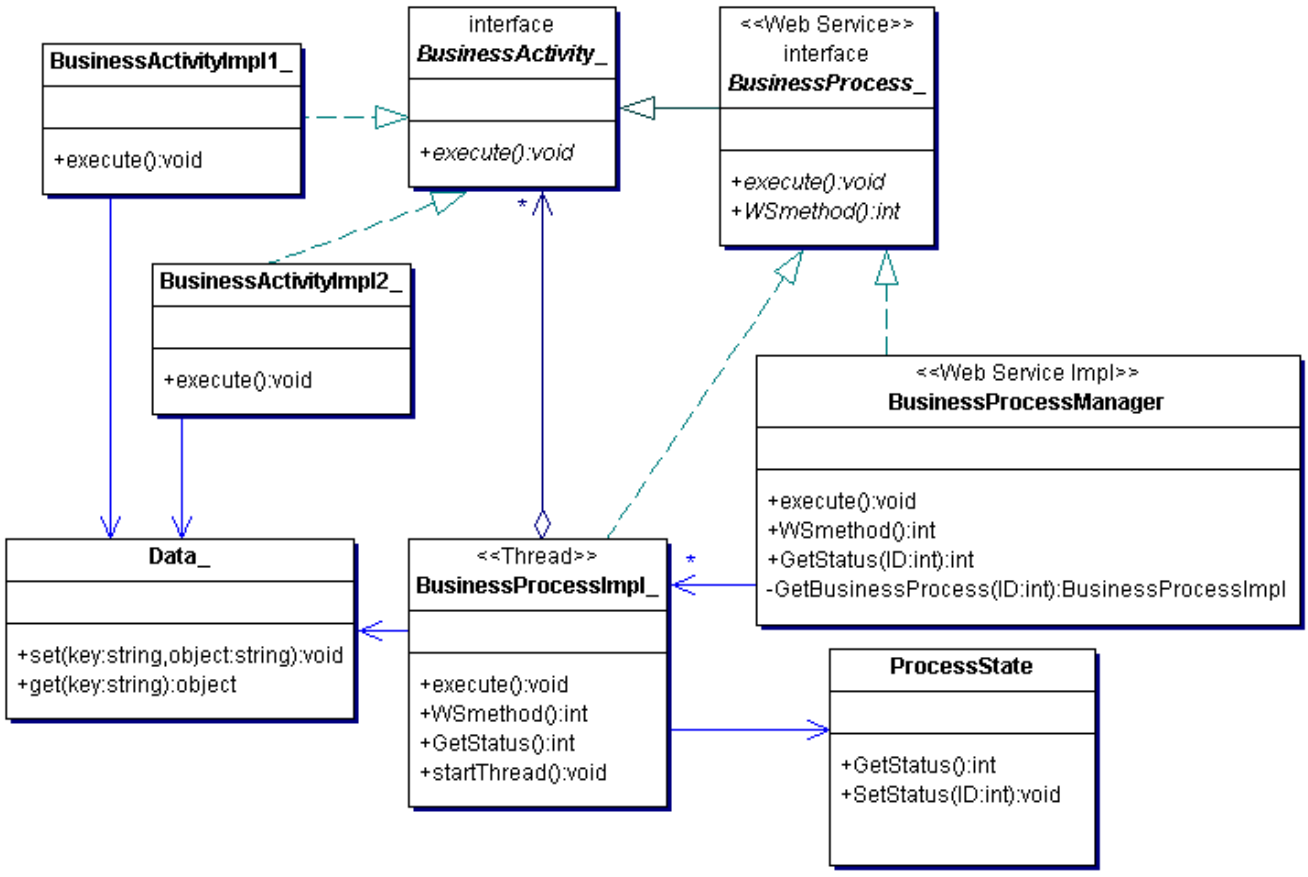
1.1.7 Solution

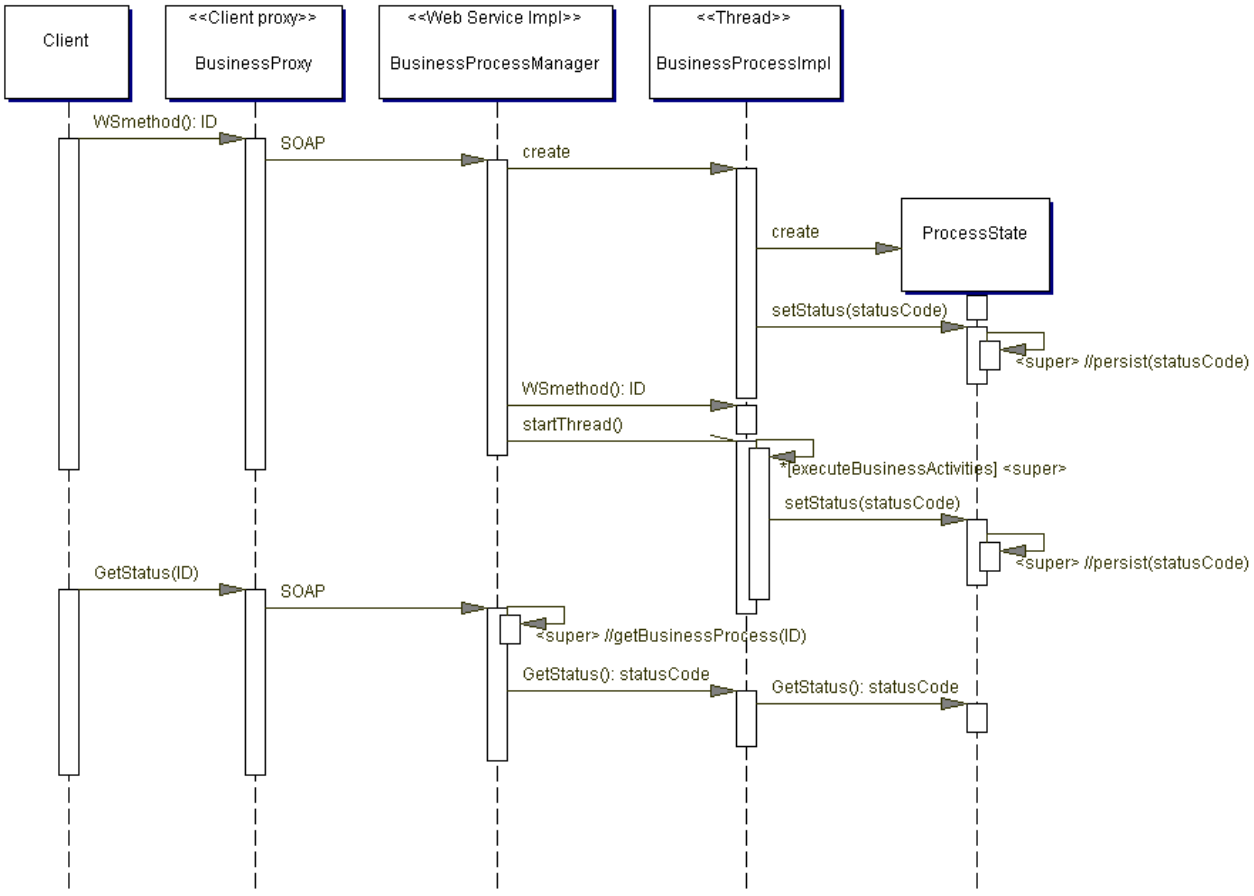
You can expose a business process manager as the web service instead of letting the long-running business process class itself be a web service. The responsibility of the manager class is to start the business process in a separate thread and quickly return a unique process identifier to the client. That kind of identifier is described in the "Correlation Identifier pattern" [EIP 03]. The manager class also provides an interface that enables the client to query the manager about the current state or progress of the business process. The state of the process should be stored persistently to enable recovery of a process, since a process can be very long-running and some of the involved servers used from the business activities may crash or for other reasons have to be restarted before the process has finished. The state can be handled by the business process itself or by some business activity or business object class, instead of using a separate process state object.

For example, a business process for ordering a product, may have the business object “Order” that also plays the role of the process state object, and it may have a status method that e.g. defines whether the order has yet been delivered or not. The identifier of the “Order” (e.g. the primary key of the order row in an RDB) might also be used as the identifier to the entire process that the client uses later for checking the status of the business process. Then the web

service process manager for the asynchronous business process can delegate a “getStatus” invocation to the “getStatus” method of the business object “Order”.

The only new or modified classes in the class diagram below, compared to the pattern “Business Process (Composition)”, are BusinessProcessManager, ProcessState and BusinessProcessImpl.





[the occurrence of "<super>" in the diagram is not intentional and will be removed in the final version]

BusinessProxy – An architecture adapter generated from WSDL.

BusinessProcessManager –A web service that starts a business process in a thread and then quickly returns a unique process identifier to the client. It also provides an interface that returns the current status of the business process when the client supplies a business process identifier. Instead of creating a thread object, as indicated in the diagram, a thread instance could be picked from a thread pool to avoid expensive object instantiation.

BusinessProcessImpl – A class that runs a business process in a thread and also provides a status method. To avoid cluttering, the diagram does not illustrate the execute messages sent to the business activities while the thread executes within the “startThread” method. These messages are illustrated in the sequence diagram for the “Business Process (Composition) pattern”.

ProcessState – A class that manages the information about the current status of the business process. This does not necessarily have to be a separate class but business objects or maybe the common data object might instead provide the status information about the business process. The state should be persistent to be able to recover the process if some of the involved servers has to be restarted before the process has completed.

1.1.8 Consequences

Use a synchronous business process if you are sure that the process will finish before the client user has lost patience about getting a response. In other words, use an asynchronous business process only if necessary, since a synchronous API makes the client programming simpler.

The client must be able to interpret the possible status codes, e.g. the web service provider may provide a class with constants used as status codes.

The business process identifier returned from the initial invocation has to be saved at the client and it could e.g. be saved as a cookie if a web browser is used as client to the web service.

For performance reasons, it is better to use a thread pool that contains reusable thread instances instead of instantiating new threads for every invocation of a business process.

A business process can include multiple external web services being called from your own business activities. Make sure you understand the identifiers and data being used by these external services and how these will affect the status of your own process.

An asynchronous business process is different from a thread of execution in a programming language since a business process can span different web services and also the length of applications, i.e. the servers can be restarted and the process should still be able to continue. Therefore a long-running business process should persist the state at some appropriate checkpoints within the process to enable recovery of the process.

1.1.9 Related patterns

The “Asynchronous Business Process pattern” (ABP) extends the “Business Process (Composition) pattern” (BPC) with the two classes BusinessProcessManager and ProcessState, though the state of the process might be maintained by the business process itself or some business activity class instead of using a dedicated process state class. Another difference is that the BusinessProcessImpl has a different role in ABP where it now starts a long running business process in another thread, while the class was exposed as a web service in the BPC pattern.

The client that executes an asynchronous business process will want to know when the business process has completed and that can be done by either a polling or a notifying mechanism and it is described with the polling patterns “Event monitor” or “Pollable Thread Manager”, or the notifying patterns “Observer” or “Notifying Thread Manager”. The “getStatus” method in the sequence diagram above is part of a polling mechanism.

Even though the “Service Activator Pattern” (SAP) [CJP 03] is a java specific pattern describing how to use JMS (Java Message Service) for invoking business services asynchronously, it is still similar to this ABP pattern for Web Services. The “ServiceActivator” in SAP is an object that is executing in a separate thread and invokes the business services/activities, i.e. it is corresponding to the “BusinessProcessImpl” object in ABP. The SAP also includes a notifying mechanism, which is not included in the ABP but instead separately described by some of the notifying patterns mentioned in the previous paragraph.