1.1 Business Process (Composition) Pattern

1.1.1 Name and Source

Business Process (Composition) pattern
Page 131-151 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 provide guidance for combining business activities into a Web Service with a well defined, large-grained and flat interface.

1.1.5 Problem

A business process is composed of one or more business activities, where each business activity also might be a business process aggregating other activities. Large business processes can involve multiple companies and it may be necessary to coordinate the business activities within the process, i.e. one activity may need input from another activity within the process.

A business process should be defined with a coarse-grained interface since you want to use few remote invocations to minimize network communication overhead. The interface should also be flat and not expose an object-oriented model since you do not want to impose dependencies to the clients of your object model. Clients also may have problem to use object-oriented models if the client language is not object-oriented. Another problem with object models is that the client objects are recreated with data sent in SOAP messages and they are not remote references. This will require extra steps in the programming to update the data back to the server, as described in the business object and business object collection patterns.

How can a flat and interoperable coarse-grained business process be designed to enable aggregation of web services and automatic execution of business activities through a generic interface, while also being able to exchange data between the activities participating in the process ?

1.1.6 Forces

1.1.7 Solution

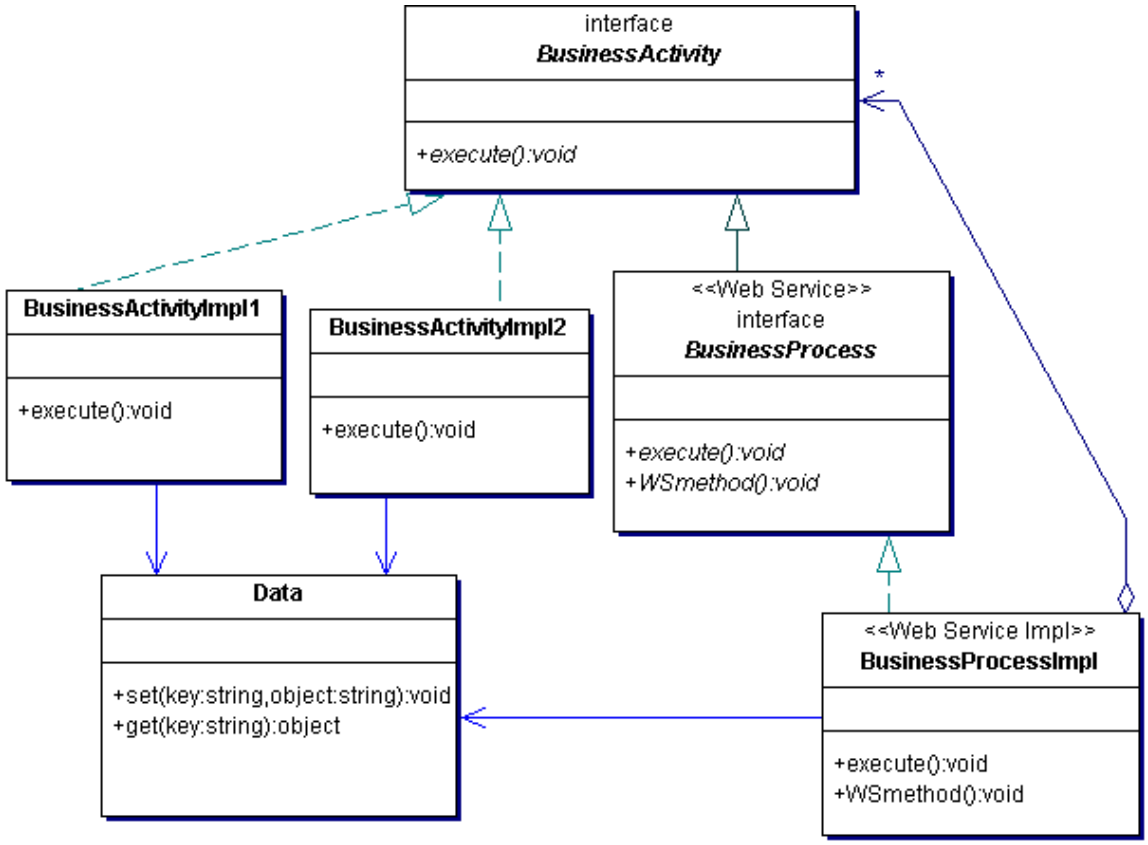
To achieve the wanted flatness of the interface you can, for example, instead of using an array of objects you can use multiple arrays with primitive data types, which of course should have the same length, and the data at the corresponding indexes should represent properties of the same object. For example, instead of defining a parameter that is an array of person objects you can use multiple parameters, e.g. one array of strings with the names, and one array of integers with the ages of the people, and so on.

To enable automatic execution of the aggregated business activities within a process, a composite (GoF) structure can be used. Such activities with a generic interface might be defined

in a declarative manner rather than programming, to then be executed within a general execution engine. The interface should provide at least one method that will execute the activity, and that method can be considered as the Execute method in the command (GoF) pattern.

The common data that needs to be shared can be implemented with a reference to a data object that is sent to the constructor of each BusinessActivity that may need it.

The elements in the diagram below do not necessarily have to be considered as classes. For example the class BusinessProcessImpl might instead be a BPEL (Business Process Execution Languages) file, which is an XML file that defines a sequence of business activities and data relationships. That file could then execute in a so-called BPEL container.



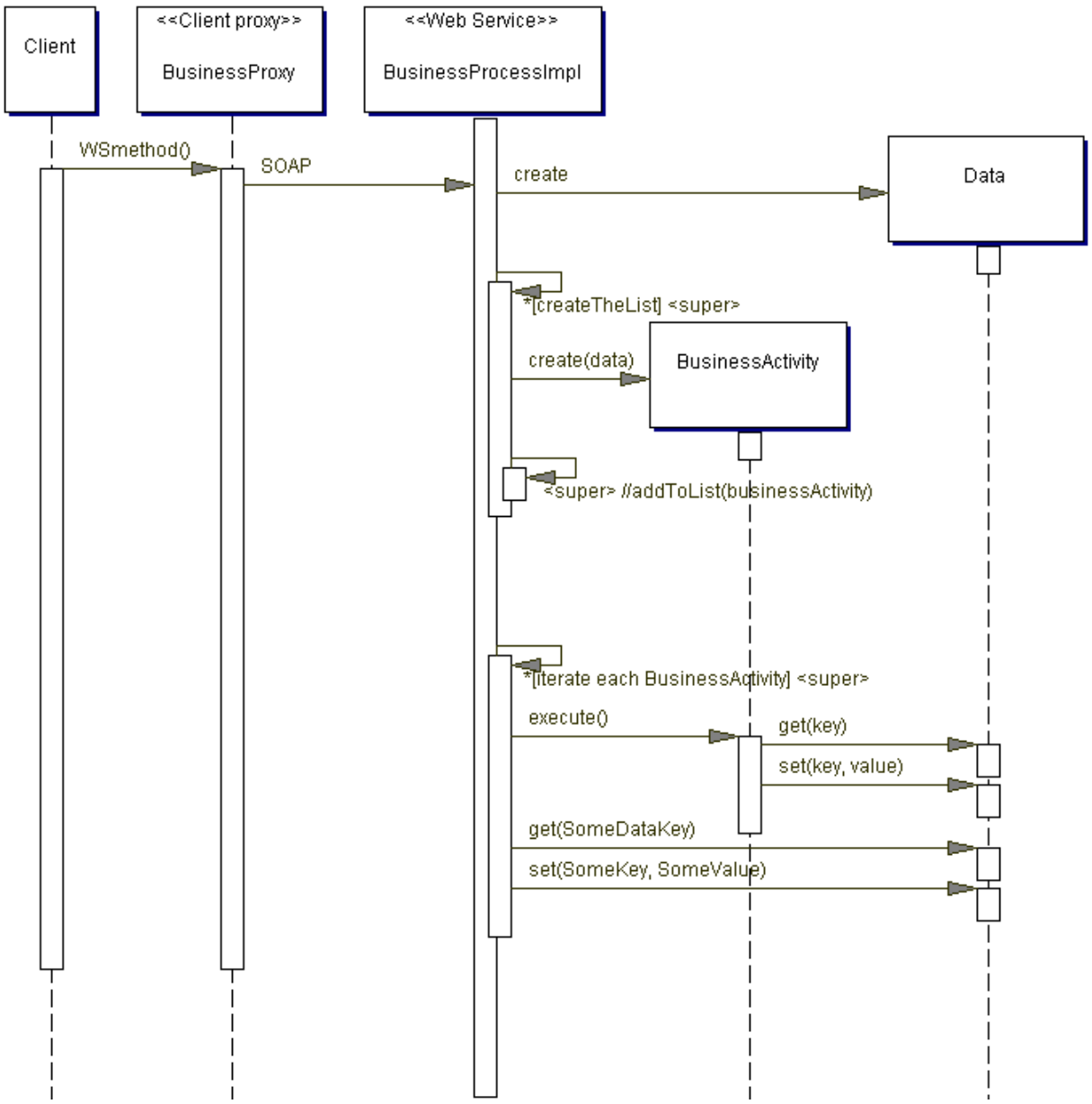
BusinessProcess – An interface generated from WSDL or extracted from a web service implementation (as in "Web Service Interface pattern"), but after that generation it probably (depending of what tool is used for generation) must be modified to also extend the BusinessActivity interface.

BusinessProcessImpl – A Web Service that implements the BusinessProcess interface by aggregating a list of business activities that together will fulfill the task of the business process. Often a business process Web Service will implement many different interfaces built for different customers. A simple implementation might do nothing more than just executing all activities, but the class also may contain much more complicated logic e.g. code that handles transactions. The choice of which next activity to execute may depend on data collected in execution of previous activities.

BusinessActivity – A business activity represents a unit of work and is the base interface for each activity or process, corresponding to the interface Component in the GoF composite pattern. It defines at least one "execute" method (GoF command) that may be executed by the BusinessProcessImpl while iterating through the list of the aggregated business activities.

BusinessActivityImpl1 & BusinessActivityImpl2 – Examples of business activities that can be included in the list of activities defined and executed by BusinessProcessImpl. Some of these activities can contain the implementation themselves while others may be delegating to other "receivers" (GoF command pattern) e.g. some activities may call other web services.

Data – The common pool of data that the business process and activities use for exchanging data. The data object can be constructed within the BusinessProcessImpl and can then be used as constructor parameter to the BusinessActivityImpl objects when these also are constructed from BusinessProcessImpl.



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

1.1.8 Consequences

Usually methods are somewhat self-documenting when you use relevant method names, but if you use a hash table for the common data object, it is important to document the keys and values that are expected to be set as preconditions to the different activities within a process.

One aspect to consider when business processes are discussed is the structure of a business process, and another aspect is how to standardize the interfaces used in business processes. This pattern only discusses the structure of a business process, and not the problem of how to standardize the interfaces used in business processes. In the future you can expect a large number of standardized business process interfaces accessible through tModels registered in UDDI.

In general, I do not believe that the common business activity interface in the pattern will be very meaningful. Business processes are usually not as simple as in the example above where all activities can execute from within an iteration. If they can, then indeed polymorphism will be useful to reuse code by invoking the “execute” method in the common interface while simply iterating the activities. However, often certain activities within a process can run in parallel processes while some activities will have to be synchronized at certain checkpoints. Branching statements (if/else or switch) may also be used for executing some activities only under certain conditions. In other words, it will typically not be possible to treat the activities uniformly and therefore the common interface will not be as useful, because when you will have to do programming for a specific class then you might just as well invoke methods with different signatures.

Business processes can be quite complicated if you are going to do the programming yourself and it is probably a good idea to learn BPEL (Business Process Execution Language) and use a graphical tool to draw the business activities in the process and let the tool generate the XML that in a declarative manner defines the execution sequence including branching and synchronization of parallel activities, and the data to be exchanged between activities. When you are finished with the BPEL XML file, you deploy it into a BPEL execution engine that then may generate classes and interfaces and/or use reflection to be able to automatically invoke the business activities as defined in the business process described by a BPEL file.

1.1.9 Related patterns

As described above, this “Business Process (Composition) pattern” can aggregate many business activities. The “Application Service pattern” [CJP 03] is another pattern that provides a coarse-grained API for aggregated behavior and coordination of multiple business objects or external services, i.e. web services.

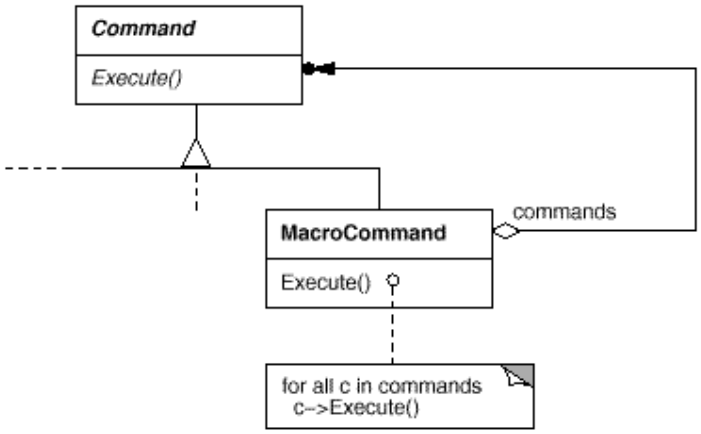
This pattern can more or less be considered as an example of the GoF macrocommand pattern, which is a combination of the GoF composite and the GoF command patterns. The business process pattern provides coordination between the different business activities (commands) through a common data object. This data object maybe could be considered as a “Receiver” object in the command pattern, although the individual commands (business activities) may also use other true receiver objects than the common data object and those receivers are typically different for the different activities. One difference though, compared with the GoF Macrocommand, is that GoF says that “MacroCommand has no explicit receiver, because the commands it sequences define their own receiver” while the data object in the Business process also can be used by the “BusinessProcessImpl” (the “MacroCommand”) which also may contain more logic than simply executing a sequence of BusinessActivities/Commands.

The GoF diagrams (macrocommand, composite and command) below shows the following corresponding classes, when compared with classes in this “Business Process (Composition) pattern”.

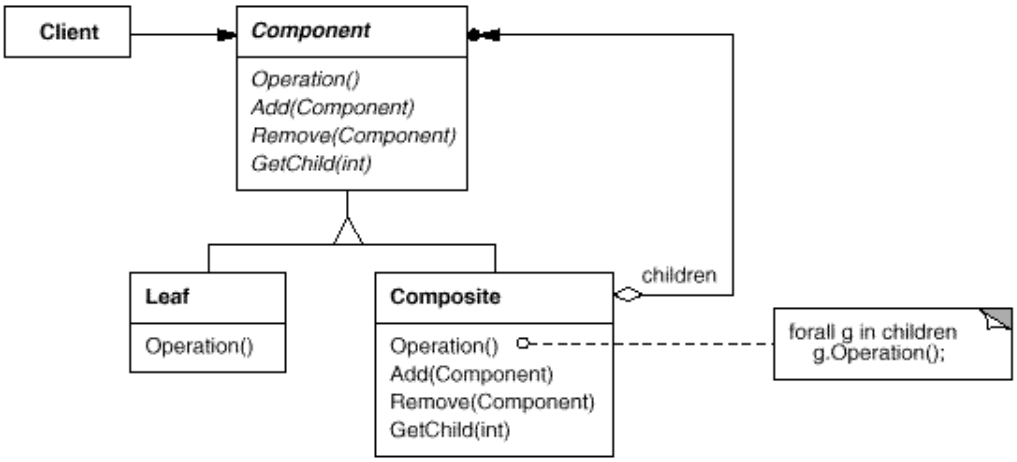
BusinessActivity = Command

BusinessProcessImpl = MacroCommand/Composite

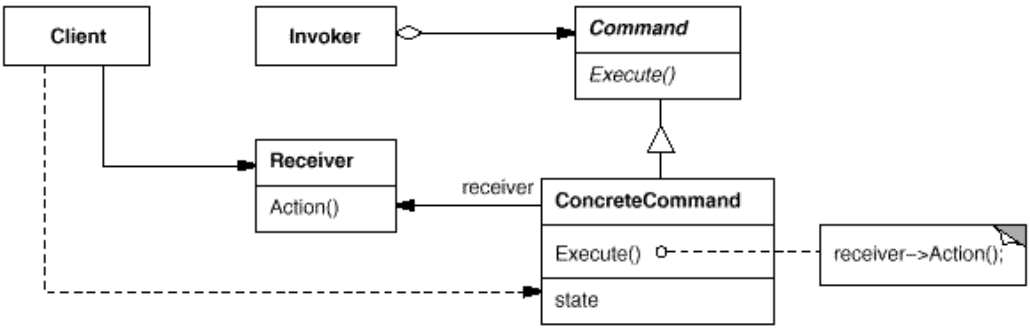
BusinessActivityImpl = ConcreteCommand/Leaf



The GoF Macrocommand pattern (Command + Composite)



The GoF Composite pattern



The GoF Command pattern