Handling Atomic Business Services

(conference version)

Geert Monsieur, Lotte De Rore,
Monique Snoeck and Wilfried Lemahieu

Katholieke Universiteit Leuven
Faculty of Business and Economics
The Leuven Institute for Research on Information Systems (LIRIS)
Naamsestraat 69, 3000 Leuven (Belgium)
firstname.lastname@econ.kuleuven.be

Abstract

This article discusses the handling of atomic business services, which are service compositions that orchestrate and coordinate underlying services to process a high-level business activity. The main contribution made in this article is the presentation of the pattern ABS handler, which describes how one can implement an atomic business service (ABS). This pattern functions as an overview pattern for a complete pattern language that is outlined in the text. This pattern language provides the appropriate ingredients for the implementation of an ABS. It is presented using thumbnails.

1 Introduction

A business process can be defined as a collection of activities that take one or more kinds of inputs and create an output that is of value to the customer [6]. These inputs are retrieved from several business applications that need to be invoked to execute the business process. In a service-based environment, business processes are supported and executed by orchestrating several services. This orchestration, sometimes referred to as service composition, can be very complex since many message exchanges between the participating services are needed.
In order to structure a business process implementation, it is a good idea to consider the business process as a set of activities in which each activity matches an invocation on a atomic business service (ABS), which is responsible for the required interactions with the services and applications.
The structuring approach can be considered as an application of MACROFLOW-
MICROFLOW described by Hentrich and Zdun [8]. This pattern structures a pro-
cess model into two kinds of processes, macroflow and microflow. The microflow
is only used for refinements of the macroflow activities. The macroflow repre-
sents the long-running, interruptible process flow, which depicts the business-
oriented process perspective. The microflow represents the short-running trans-
actional flow that depicts the IT-oriented process perspective.

Another similar approach can be found in the many studies about Web ser-
vice conversations [1, 12, 3, 2, 7]. Roughly speaking, conversations can be
considered as well-described (low-level) interactions between specific services
(partners), which can be used as a kind of module in a high-level process.

Finally, the research about interactions protocols or business protocols by De-
sai and Singh are certainly worth to mention [5, 11]. Similar to the conversation-
based method the concept of busines protocols is presented as a (design) abstrac-
tion that provides information about which sequence of message exchanges or
interactions can occur between several parties. However, they argue that busi-
ness processes are conventionally modeled directly as monolithic flows. They
state that these flows are often more complex than necessary and lack modu-
ularity [5]. Therefore they propose to model business processes as composition
of business protocols. Directly developing composite business flows is harder
than modeling individual business protocols and then putting them together.
Thus, protocol-based process modeling can be viewed as a structured approach
wherein protocols are granules [5].

As an ABS is in its turn a composition of a set of services delivered by one
or (usually) more components, the orchestration of these underlying business
component services will be the "microflow", which represents the short-running
transactional flow. This microflow defines the way the individual business com-
ponent services are composed and coordinated to deliver the required ABS.
These business component services are the ABS participants. The atomic busi-
ness service is transactional in the sense that either it is delivered completely,
or it is not executed at all. The microflow gives the technical perspective of
handling an ABS, while the ABS gives the more abstract business oriented per-
spective of a single business task. The macroflow defines the business process as
a sequence of such atomic business tasks, each of which is supported by an ABS.

The main contribution made in this article is the presentation of the pattern
ABS Handler, which describes how one can implement an ABS (see section 2).
This pattern functions as an overview pattern for a complete pattern language
that is outlined in the text. This pattern language provides the appropriate
ingredients for the implementation of an ABS. It is presented using thumbnails
(section 3).
2 Pattern: ABS handler

2.1 Context

In a service-based environment, a business process is supported and executed by orchestrating several services. In order to structure the business process implementation, each activity of the business process is matched with the invocation of an ABS. This ABS is in its turn a composition of a set of services delivered by one or (usually) more components. These components will typically be business logic components that offer services to manage (create, modify, end) business objects.

2.2 Problem

The atomic business service consists of, on the one hand, a set of rules set by the several services that must be checked and, on the other hand, a set of activities that must be executed. The use of the term atomic means that an ABS is executed successfully if all rules are satisfied and all specified activities are executed successfully. This means, when one of the rules is not satisfied or one of the activities could not be executed completely, the ABS should be rolled back as if nothing has happened.

These requirements of an ABS raise the question how to implement the handling of an ABS properly.

More particular, how do you define the microflow (i.e. which message exchanges are required?) and who is responsible for monitoring the correct execution of the microflow (i.e. which component is responsible for coordinating the microflow?).

If a company’s information systems run on only one single platform it is quite easy to coordinate all actions required to execute a business activity. In such cases most of the time no specific message exchanges are needed to communicate. Business functions are directly invoked on the business applications. If a service-oriented approach is followed, calls to components are often still synchronous, which simplifies the implementation of an ABS. Unfortunately, in order to support a company’s business processes in an optimal way, each department in a company often has their own specific business applications, which should be integrated in an efficient and effective manner. Besides the intra-organizational integration of business applications today’s businesses require intra-organizational integration of information systems. In these cases business processes are executed on a distributed platform. This demands a service-oriented approach in which business processes are executed by combining an appropriate set of services. In this setting, services (often implemented as Web services) are spread out across different locations and companies. This makes the implementation of ABSs difficult. Due to the distributed nature of the information systems all communication is often asynchronous and a business transaction can take a long time in certain circumstances. Therefore verifying rules, executing activities and guarding the atomicity of ABSs is not a trivial
thing to do.
In summary one can say that implementing ABSs is quite easy to do in limited conditions, e.g. a single platform. Once participating components are spread out on a distributed system it becomes a challenge to maintain the properties of ABSs.

### 2.3 Example

The business process for processing orders in an online shop consists of several main business activities, e.g. *create order*, *process order*, *invoice* and *pay*. These business activities need to be implemented as ABSs. The microflow of the *create order* ABS consists of interactions with the following services: the sales and marketing service, the finance service and the stock management service (see figure 1). Besides these services other services as e.g. the customer support service or a shipping service could be part of the company’s information systems.

In order to handle the *create order* ABS, it is required that these three services (sales and marketing, finance and stock management) are invoked. Therefore, all these participating services will set certain restrictions or business rules on the *create order* business activity. For example, relevant business rules in the case of *create order* could be 'checking if the customer is creditworthy' (information controlled by finance service), 'checking if the customer’s order doesn’t exceed the maximum allowed amount to order' (information controlled by the sales and marketing service) and 'checking if there is enough in stock' (information controlled by the stock management service). All these business rules need to be checked before any action related with the *create order* activity can be undertaken in one of the participating services. This is important to maintain the atomicity of the ABS *create order*. This means that either all business activities in the participating services are completely and successfully executed, or none participating service has executed the business activity. For *create order* this means participating services process this activity only if the customer is creditworthy, the desired amount is in stock and the maximum allowed amount to order is not reached.
2.4 Forces

- **Dependencies:** The correct handling of an ABS requires some coordination. The responsibility of this coordination task could be assigned to one component (one of the participants or a separate, new component), or it could be distributed amongst several components. The way coordination is assigned to components creates dependencies between these components. In order to maximize the adaptability of the implementation architecture, it is recommended to minimize dependencies as much as possible.

- **Confidentiality of business rules:** The participating services related to an ABS all set some restrictions or business rules on the processing of a (high-level) business activity. Before processing and executing the business activity in all participating services these rules need to be checked. Business rules set by all participating services can be shared and known publicly. However some participating services are not willing or simply cannot share all business rules. In the case of the create order business activity one can imagine that the finance service cannot simply share the rules used to check the creditworthiness of customers. In particular it can be possible that the finance service relies on other external services (e.g. the customer’s bank, the credit card company, etc.) to check the creditworthiness of a customer. In that case it is difficult to share the rules.

In a business-to-business environment it is common that business partners do not share all business rules, since they want to be sure these rules never arrive in hands of competitors. Notice that in other cases (e.g. the maximum amount allowed to order) business rules can be quite easily shared.

- **Variability:** An ABS is a composition of several services, which are referred to as the participating services. In the case of create order business activity three participating services are given. One could think of business cases in which it should be possible to add more services as participating services at run time (e.g. adding a shipping service as participating service for the create order business activity).

- **Technical capabilities of the participating services:** There exists probably many strategies and scenarios to handle an ABS. However, sometimes not all scenarios are always supported by the participating services. It all depends on the interface of the participating services, which should provide the appropriate actions needed for a specific ABS handling strategy.

- **Required performance:** The way the handling of an ABS is implemented can have substantial impact on performance. When implementing the handling of an ABS it can be important to keep in mind that a certain level of performance is required.
2.5 Solution

To minimize dependencies between components it is advisable to create a separate ABS handler. This handler is responsible for the coordination of the microflow and guards the atomicity of the ABS. Considerations while implementing the ABS handler are on the one hand the use of a subscription manager and on the other hand the coordination protocol used.

The subscription manager is a component that stores information about which services are participants in an ABS. By means of this component it is also possible for services to dynamically subscribe for certain ABSs. If no subscription manager is used in the handling of an ABS it is required to hardcode all participating services that needed to be coordinated. It should be clear that the choice between using a subscription manager or hardcoded all participating services is often a trade-off between on the one hand the environmental variability, and on the other hand the performance demanded by the business case.

The last component in the solution for the ABS handling problem is the coordination protocol used. This protocol defines the steps that the ABS handler (the coordinator) should follow to implement the coordinated handling of an ABS. The two-phase commit coordination protocol consists of two main phases or steps. In the first step the business rules set by all the participating services are checked. If the results of this checking phase are positive the protocol continues with the second step, which consists of instructing the participating services to execute the corresponding activities to finish the processing of the atomic business service. It is assumed that the results of the business rules checks are still valid when instructing the participants in the second phase. This probably implies that the participating services should lock some resources. Since locking resources in a distributed and service-oriented environment can be quite expensive when dealing with long-during transactions, the two-phase commit does not suffice in every scenario. Therefore another possible coordination protocol is the compensation-based coordination protocol. Instead of locking resources services in a first step the handler instructs the participating services to execute the business activity at once. If one or more services fail to execute the business activity successfully because of business rule violation or other problems the coordinating component is responsible for the compensation of the actions undertaken in all participating services.

If the ABS handler needs to know if there are some business rules violated it has two options to choose from. Either the participating services share the business rules with the handler and rules are checked centrally, or the participating services check the business rules themselves and the results are sent back to the handler. In the first option business rules are kept private and checked decentrally, while in the second option business rules are shared with the ABS handler and checked centrally. Choosing between these two op-
tions is probably much easier when considering the required performance and confidentiality of business rules. If performance is far more important than the confidentiality of business rules, one is advised to go for the option in which business rules are shared with the ABS handler and checked centrally. In the other case, when confidentiality is more important than performance, it is better to keep business rules private and checked decentrally. In addition to these two coordination protocols there exists many other transaction coordination protocols, which are in many cases minor variations on the protocols discussed above.

Figure 2 gives an overview of the pattern language. The relationship between ABS handler and all other patterns is shown.

2.6 Consequences

- By putting the responsibility of handling ABSs in a separate component, firstly the basic services can remain independent of each other and secondly the handlers can be put in a separate layer on top of the business component layer, creating a dependency from handlers toward business components, but not the other way around.

- Using the private and checked decentrally pattern implies that business rules are kept confidential, while shared with the ABS handler and checked centrally means that a central component should have knowledge of all business rules and thus business rules are shared.

- In the case of a huge variability in participating services, an ABS han-
DLER should be implemented by means of a look-up table, which allows services to subscribe at run time. The use of a SUBSCRIPTION MANAGER guarantees a high level of flexibility in case of variable business cases. While on the other hand in a stable environment it can be interesting (e.g., due to performance reasons) to hard code the participating services in an ABS HANDLER.

- In the solution we used two possible coordination protocols: TWO-PHASE COMMIT COORDINATION PROTOCOL and COMPENSATION-BASED COORDINATION PROTOCOL. Using the TWO-PHASE COMMIT COORDINATION PROTOCOL implies the availability of appropriate actions to check business rules at the participating services.

- Performance is also very much linked to scalability. If you sell a few tens of products every day, it is probably not so much of a problem if one generic ABS HANDLER is responsible for all ABSs. On the other hand, if thousands of orders come in per day, then performance and scalability to large volumes of transactions becomes a real issue. In the latter case it can be important to make several ABS HANDLERS instead of one ABS HANDLER which can handle more than one atomic business services. Notice that the use of a SUBSCRIPTION MANAGER increases the flexibility of the ABS HANDLER to allow variability in participating services, but it also reduces the performance of the overall system since many lookup actions are needed.

2.7 Example resolved

- Minimized dependencies: Suppose that the handling of the create order ABS is handled by the Sales and Marketing components. Then this component becomes dependent on the finance component and the stock management component. Other ABSs may create other dependencies. For example, the coordination of the ABS invoice could be assigned to the Finance component, creating a dependency in the other direction between the Finance and the Sales and Marketing component, as the order amount and conditions for discount rules should be checked there. Therefore it is useful to create a separate ABS HANDLER which deals with the handling of one or more atomic business activities.

- Dealing with confidential business rules: As the finance service is a participant for the create order ABS it sets some business rules on this business activity. In particular it is the responsibility of the finance service to check if the customer is creditworthy. Since the specific business rules used to check this issue cannot be made public (e.g., because the finance relies on other external services to check the creditworthiness of customers) it is required to check the business rules decentrally for the create order ABS (PRIVATE AND CHECKED DECENTRALLY).
• **Room for variability:** Since in the example it should be possible to add a shipping service, which can happen quite frequently, it is desirable to have a flexible handling system which allows a sufficient variability in participating services. Therefore the use of **subscription manager** can add value to the handling of the *create order* **ABS**.

• **Dealing with technical capabilities:** Suppose the stock management service does not provide the necessary interfaces to check the stock level of a given product or to check there is enough in stock to process a given order. So suppose the interface only allows to directly execute an order in the stock management service. If the service finds out the stock level is not sufficient an error is returned. This makes it difficult or even impossible to go for the **two-phase coordination protocol**. Therefore the **ABS handler** for *create order* needs another coordination protocol (e.g. a **compensation-based protocol**).

### 2.8 Known uses

In [4] a prototype of a Web service orchestration layer overlaying Web Service Description Language (WSDL)\(^1\) and Business Process Execution Language (BPEL)\(^2\) was developed using the pattern language for handling ABSs. The pattern language presented in this article was also applied in a case study, which dealt with the integration of COTS\(^3\) applications for customer management, finance and service provisioning for a Dutch broadband provider [9]. In the remainder of this subsection we outline the case study. More details can be found in [9, 10].

We consider a business process for order handling in the telecommunication company. Executing the business process is done by interacting with four COTS applications (Sales and Marketing, Service Provisioning, Finance and Customer Support). Since defining business processes as sequence constraints on message exchanges is considered a too low-level task, the business process at hand is described using ABSs. These ABSs are responsible for the interactions required to execute a business activity. Figure 3 gives a high-level overview of the business process using four ABSs (*create order*, *install*, *invoice* and *pay*). The company needed a solution which was scalable, flexible and implementable in the current environment, consisting mainly of COTS applications. Several forces (as mentioned in 2.4) made it complex to design an appropriate solution: e.g. sharing business rules was difficult when using certain COTS applications, some COTS applications have limited support for certain coordination protocols, etc. These forces made it difficult to choose between several implementation alternatives. As such, the **ABS handler** pattern helped to make the right decisions. Furthermore it provided clear descriptions of possible solution aspects (by means of the subpatterns).

---

\(^1\)http://www.w3.org/TR/wsdl

\(^2\)http://www.oasis-open.org/committees/wsbpel

\(^3\)Commercial off-the-shelf
3 Thumbnails

- **Subscription Manager**
  This is a component or service which allows to store information about ABSs and corresponding participants. It also allows services to dynamically subscribe and unsubscribe for certain ABSs.

- **Hardcode All Participating Services**
  Instead of using a Subscription Manager it is also possible to hardcode all participating services in the ABS Handler.

- **Two-Phase Commit**
  This coordination protocol consists of two phases. In the first phase the processing of an ABS is prepared. If the first step ended successfully the processing of an ABS is completed in the second phase.

- **Compensation Based**
  All participating services try to process an ABS. If any participant fails to do so all services are asked to compensate the processing of the ABS.

- **Private and Checked Decentrally**
  A participating service does not share business rules. It is only possible to ask to the participant whether or not business rules are violated.

- **Shared With the ABS Handler and Checked Centrally**
  A participating service shares business rules. As such other components (e.g. the ABS Handler) can check business rules set by the participant themselves (without communicating with the participant).

### Acknowledgements

The authors would like to thank Alejandra Garrido for her many constructive and valuable remarks during the shepherding process. This has been written as
part of a project funded by the Research Fund K.U.Leuven (OT 05/07), whose support is gratefully acknowledged.

References


