Handling Transactional Business Services

Geert Monsieur, Lotte De Rore, Monique Snoeck, Wilfried Lemahieu
Katholieke Universiteit Leuven
Faculty of Business and Economics
The Leuven Institute for Research on Information Systems (LIRIS)
firstname.lastname@econ.kuleuven.be

ABSTRACT

This article discusses the handling of transactional 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 TBS HANDLER, which describes how one can implement a transactional business service (TBS). 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 a TBS. It is presented using thumbnails.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*Patterns*

General Terms

Design, Management, Languages

Keywords

Business Process, Transactional Business Service, Service Composition

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 *transactional business service* (TBS), 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 process model into two kinds of processes, macroflow and microflow. The microflow is only used for refinements of the macroflow activities. The macroflow represents the long-running, interruptible process flow, which depicts the business-oriented process perspective. The microflow represents the short-running transactional flow that depicts the IT-oriented process perspective.

Another similar approach can be found in the many studies about Web services *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 Desai 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) abstraction that provides information about which sequence of message exchanges or *interactions* can occur between several parties. However, they argue that business processes are conventionally modeled directly as monolithic flows. They state that these flows are often more complex than necessary and lack modularity [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].

A TBS is a composition of a set of services. The orchestration of these underlying services will be the "microflow", which represents the short-running transactional flow. This microflow defines the way the individual services are composed and coordinated to deliver the required TBS. These services are the *TBS participants.* The *transactional* business 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 a TBS, while the TBS gives the more abstract business oriented perspective of a single business task. The macroflow defines the business process as a sequence of such transactional business tasks, each of which is supported by a TBS.

The main contribution made in this article is the presentation of the pattern TBS HANDLER, which describes how one can implement a TBS (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 a TBS. It is presented using thumbnails (section 3).

2. PATTERN: TBS 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 a TBS. This TBS is a composition of a set of services. These services will typically be some kind of business logic components that offer services to manage (create, modify, end) business objects.

2.2 Problem

The transactional 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 *transactional* means that a TBS 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 TBS should be rolled back as if nothing has happened.
These requirements of a TBS raise the question how to implement the handling of a TBS 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 a TBS.
Unfortunately, in order to support a company's business processes in an optimal way, each department in a company often has its 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 TBSs 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 transactional aspect of TBSs is not a trivial thing to do.
In summary one can say that implementing TBSs 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 TBSs.

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 TBSs. The microflow of the *create order* TBS 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* TBS, 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 transactional aspect of the TBS *create order*. This means that either all business activities in the participating services are completely and successfully executed, or none of the participating services 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 a TBS requires some coordination. The responsibility of this coordination task could be assigned to one service (one of the participants or a separate, new service), or it could be distributed amongst several services. The way coordination is assigned to services creates dependencies between these services. 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 a TBS 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

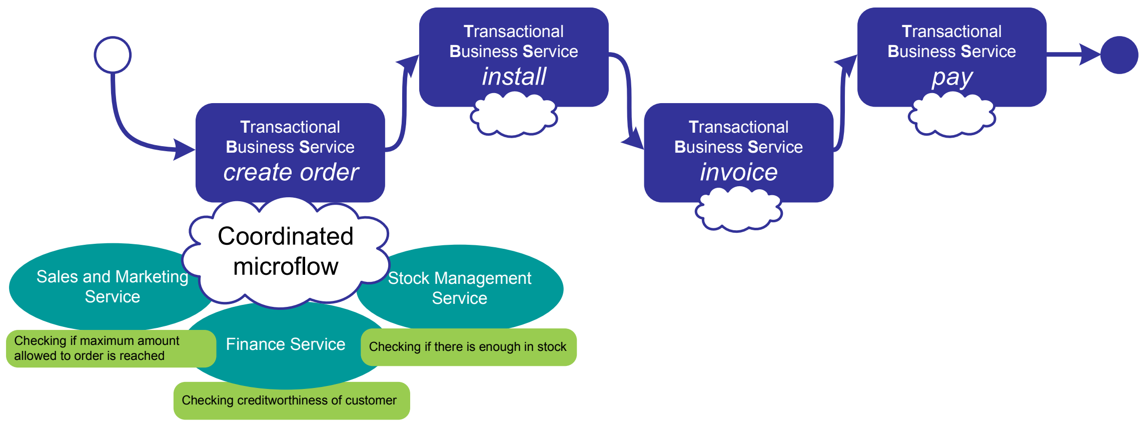

Transactional Business Service create order
Transactional Business Service install
Transactional Business Service invoice
Transactional Business Service pay
Coordinated microflow
Sales and Marketing Service
Stock Management Service
Finance Service
Checking if maximum amount allowed to order is reached
Checking if there is enough in stock
Checking creditworthiness of customer

Figure 1: Transactional Business Service: create order

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:** A TBS 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 exist probably many strategies and scenarios to handle a TBS. However, sometimes not all scenarios are 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 TBS handling strategy.

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

2.5 Solution

To minimize dependencies between services it is advisable to **create a separate TBS handler. This handler is responsible for the coordination of the microflow and guards the the transactional aspect of the TBS**. Considerations while implementing the TBS HANDLER are on the one hand how to manage the different participating services and on the other hand the coordination protocol used.

The SUBSCRIPTION MANAGER is a component that stores information about which services are participants in a TBS. By means of this component it is also possible for services to dynamically subscribe for certain TBSs. If no SUBSCRIPTION MANAGER is used in the handling of a TBS 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 HARDCODING 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 TBS handling problem is the coordination protocol used. This protocol defines the steps that the TBS HANDLER (the coordinator) should follow to implement the coordinated handling of a TBS. 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 transactional 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 implies that the participating services should probably lock some resources. Since locking resources in a distributed and service-oriented environment can be quite expensive when dealing with long-running 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 requesting the participating

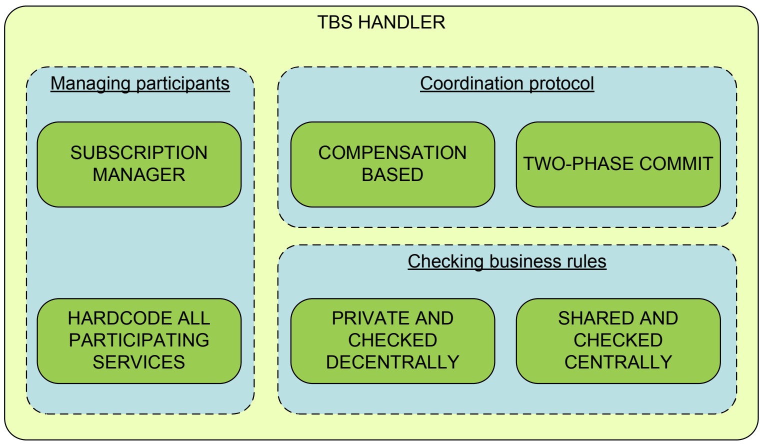


Figure 2: A pattern language for handling a TBS

services to compensate the actions undertaken. In addition to these two coordination protocols there exist many other transaction coordination protocols, which are in many cases minor variations on the protocols discussed above.
If the TBS 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 TBS HANDLER AND CHECKED CENTRALLY. Choosing between these two options 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 TBS 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.

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

2.6 Consequences

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

- Using the PRIVATE AND CHECKED DECENTRALLY pattern implies that business rules are kept confidential, while SHARED WITH THE TBS HANDLER AND CHECKED CENTRALLY means that a central component should have knowledge of all business rules and thus business rules are shared.

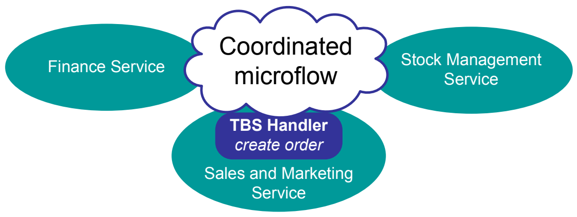
- In the case of a huge variability in participating services, a TBS HANDLER should be implemented by means of a look-up table, which allows services to subscribe at run time. The use of a SUBSCRIPTION MANAGER guarantees a high level of flexibility in case of variable business cases. While on the other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

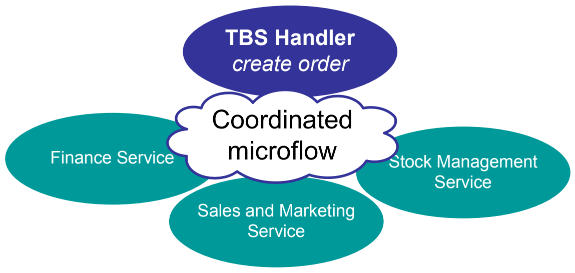
- Performance is also very much linked to scalability. If you sell a few tens of products every day, it is probably not so much of a problem if one generic TBS HANDLER is responsible for all TBSs. On the other hand, if thousands of orders come in per day, then performance and scalability to large volumes of transactions becomes a real issue. In the latter case it can be important to make several TBS HANDLERs instead of one TBS HANDLER which can handle more than one transactional business service. Notice that the use of a SUBSCRIPTION MANAGER increases the flexibility of the TBS 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* TBS is handled by the Sales and Marketing service (see figure 3(a)). Then this service becomes dependent on the finance service and the stock management service. Other TBSs may create other dependencies. For example, the coordination of the TBS invoice could be assigned to the Finance service, creating a dependency in the other direction between the Finance and the Sales and Marketing service, as the order amount and conditions for discount rules should be checked there. Therefore it is useful to create a separate TBS HANDLER which deals with the

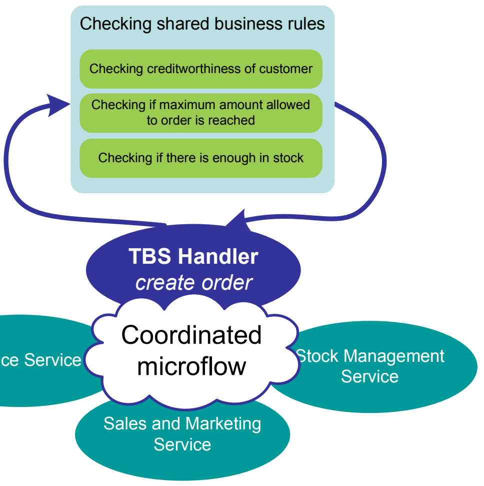


(a) No use of separate TBS handler

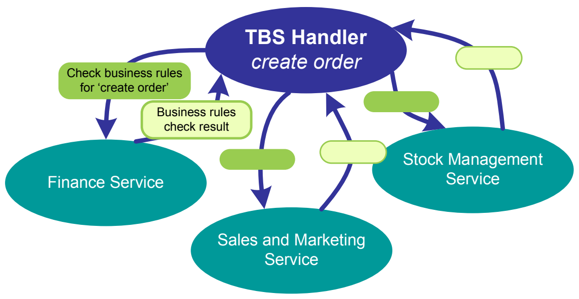


(b) Separate TBS handler

Figure 3: Minimizing the dependencies by adding a separate TBS handler

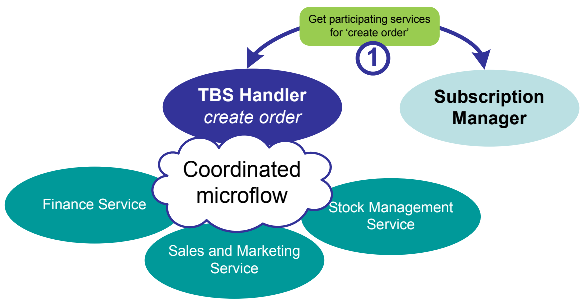

Checking shared business rules
Checking creditworthiness of customer
Checking if maximum amount allowed to order is reached
Checking if there is enough in stock
TBS Handler
create order
Coordinated microflow
Finance Service
Stock Management Service
Sales and Marketing Service

(a) Business rules are shared

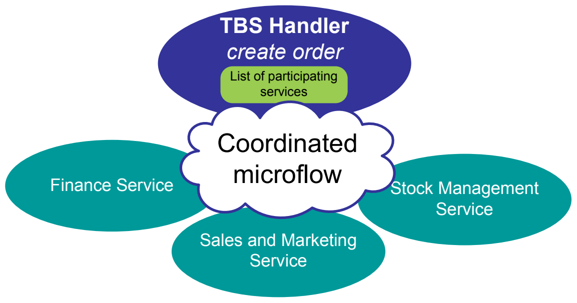

TBS Handler
create order
Check business rules for 'create order'
Business rules check result
Finance Service
Stock Management Service
Sales and Marketing Service

(b) (Finance) Business rules are private

Figure 4: Dealing with the confidentiality of business rules

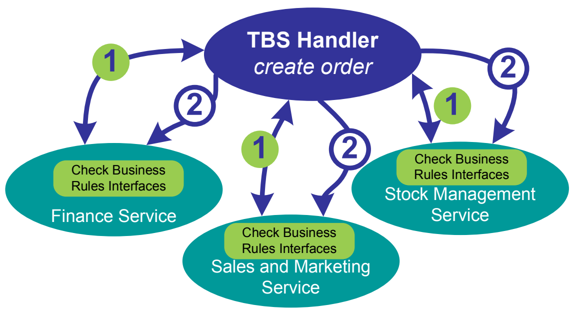


(a) A Subscription Manager holds list of participating services

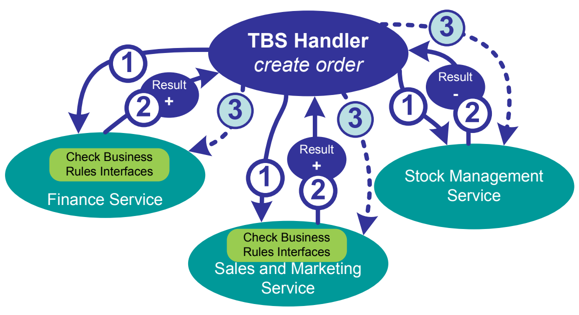


(b) Participating services are hardcoded in the TBS handler

Figure 5: Dealing with variability


TBS Handler
create order
1
2
2
1
1
2
Check Business Rules Interfaces
Finance Service
Check Business Rules Interfaces
Sales and Marketing Service
Check Business Rules Interfaces
Stock Management Service

(a) All TBS participants have interfaces for checking business rules. Step 1 consists of checking business rules. Step 2 is about processing the high-level business activity.



(b) Since the Stock Management Service does not support checking business rules, step 1 consists of processing the high-level business activity. Depending on the result (see result arrows in step 2), it is possible that compensating actions are needed (see dashed arrows - step 3).

Figure 6: Dealing with the technical capabilities of services

handling of one or more transactional business activities (see figure 3(b)).

- **Dealing with confidential business rules:** As the finance service is a participant for the *create order* TBS 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 service relies on other external services to check the creditworthiness of customers) it is required to check the business rules decentrally for the *create order* TBS (PRIVATE AND CHECKED DECENTRALLY) (see figure 4(b)).

- **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* TBS (see figure 5(a)).

- **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 if 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 (see figure 6(a)). Therefore the TBS HANDLER for *create order* needs another coordination protocol (e.g. a COMPENSATION-BASED PROTOCOL) (see figure 6(b)).

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 TBSs.

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 TBSs. These TBSs are responsible for the interactions required to execute a business activity. Figure 7 gives a high-level overview of the business process using four TBSs (*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 TBS HANDLER pattern helped to make the right decisions. Furthermore it provided clear descriptions of possible solution aspects (by means of the subpatterns).

3. THUMBNAILS

- SUBSCRIPTION MANAGER
  This is a component or service which allows to store information about TBSs and correspondig participants. It also allows services to dynamically subscribe and unsubscribe for certain TBSs.

- HARDCODE ALL PARTICIPATING SERVICES
  Instead of using a SUBSCRIPTION MANAGER it is also possible to hardcode all participating services in the TBS HANDLER.

[1]http://www.w3.org/TR/wsdl
[2]http://www.oasis-open.org/committees/wsbpel

[3]Commercial off-the-shelf

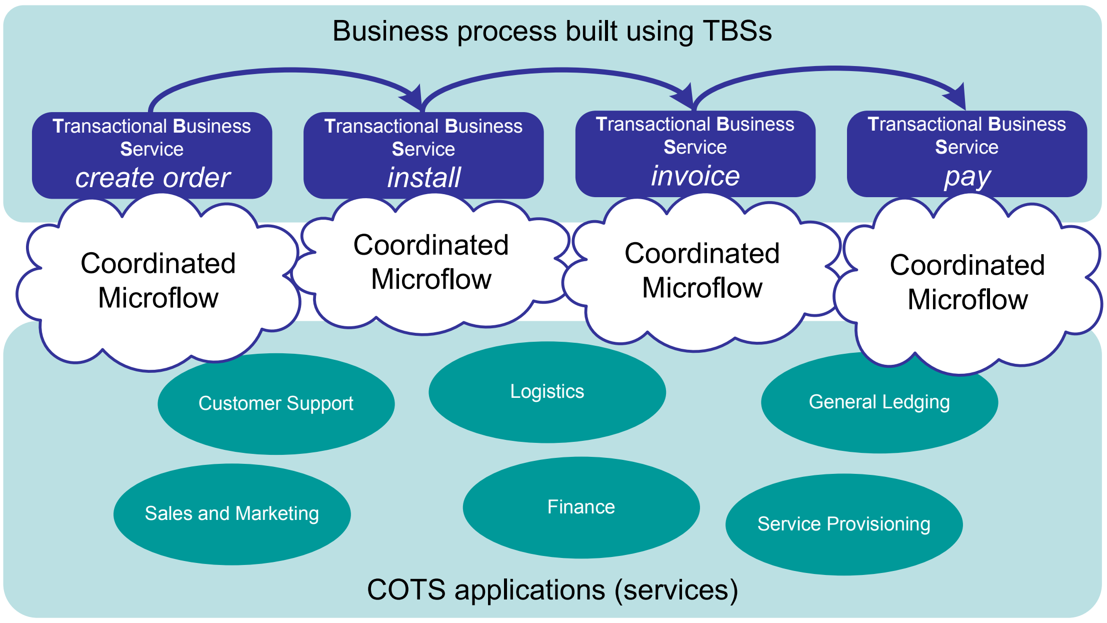

Business process built using TBSs
Transactional Business Service create order
Transactional Business Service install
Transactional Business Service invoice
Transactional Business Service pay
Coordinated Microflow
Coordinated Microflow
Coordinated Microflow
Coordinated Microflow
Customer Support
Logistics
General Ledging
Sales and Marketing
Finance
Service Provisioning
COTS applications (services)

Figure 7: A business process in a telecommunication company

- TWO-PHASE COMMIT
  This coordination protocol consists of two phases. In the first phase the processing of a TBS is prepared. If the first step ended successfully the processing of a TBS is completed in the second phase.

- COMPENSATION BASED
  All participating services try to process a TBS. If any participant fails to do so all services are asked to compensate the processing of the TBS.

- 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 TBS HANDLER AND CHECKED CENTRALLY
  A participating service shares business rules. As such other components (e.g. the TBS 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. Furthermore the quality of this article has increased substantially after the conference's writers' workshop. The authors thank the workshop participants, Paul Austrem, Philipp Bachmann, Rob Daigneau, Lise Hvatum, John Liebenau, Mark Mahoney, Miyuko Naruse and Bobby Woolf, for their extensive, constructive, helpful and inspiring discussion of a previous version of this article.
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.

4. REFERENCES

[1] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy, et al. Web Service Choreography Interface (WSCI) 1.0. *Standards proposal by BEA Systems, Intalio, SAP, and Sun Microsystems*, 2002.

[2] B. Benatallah, F. Casati, and F. Toumani. Web service conversation modeling: a cornerstone for e-business automation. *Internet Computing, IEEE*, 8(1):46–54, 2004.

[3] F. Daniel and B. Pernici. Insights into Web Service Orchestration and Choreography. *International Journal of E-Business Research*, 2(1):58–77, 2006.

[4] J. Decroos. The development of an event layer for Web service orchestration. *Master thesis K.U.Leuven, Faculty of Business and Economics, Belgium*, 2005.

[5] N. Desai and MP Singh. Protocol-based business process modeling and enactment. *Web Services, 2004. Proceedings. IEEE International Conference on*, pages 35–42, 2004.

[6] M. Hammer and J. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution.* HarperCollins, 2003.

[7] JE Hanson, P. Nandi, and S. Kumaran. Conversation support for business process integration. *Enterprise Distributed Object Computing Conference, 2002. EDOC'02. Proceedings. Sixth International*, pages 65–74, 2002.

[8] C. Hentrich and U. Zdun. Patterns for process-oriented integration in service-oriented architectures. *Proceedings of 11th European Conference on Pattern Languages of Programs (EuroPlop 2006)*, 2006.

[9] W. Lemahieu, M. Snoeck, and C. Michiels. Integration of third-party applications and web clients by means of an enterprise layer. *Annals of cases on information*

technology, pages 213–233, 2003.

[10] Geert Monsieur, Monique Snoeck, and Wilfried Lemahieu. Coordinated web services orchestration. *Proceedings of the IEEE International Conference on Web Services (ICWS)*, pages 775–783, 2007.

[11] M.P. Singh, A.K. Chopra, N. Desai, and A.U. Mallya. Protocols for processes: programming in the large for open systems. *ACM SIGPLAN Notices*, 39(12):73–83, 2004.

[12] X. Yi and K.J. Kochut. Process composition of web services with complex conversation protocols: a colored petri nets based approach. *Proc. of the Design, Analysis, and Simulation of Distributed Systems Symposium at Adavanced Simulation Technology Conf*, pages 141–148, 2004.