Managing data dependencies in service compositions

Geert Monsieur[a,*], Monique Snoeck[a], Wilfried Lemahieu[a]

[a]*The Leuven Institute for Research on Information Systems (LIRIS)*
Faculty of Business and Economics, Katholieke Universiteit Leuven
Naamsestraat 69, 3000 Leuven, Belgium

Abstract

Composing services into service-based systems requires the design of coordination logic, which describes all service interactions realising the composition. Coordination can be defined as the management of dependencies; in a services context we can discriminate between 'control flow' that manages sequence dependencies and 'data flow' for managing data dependencies. Current research fails to address the management of data dependencies in a systematic way and mostly treats it as subordinate to sequence dependencies. In this article a 'data flow' pattern language is presented that provides a systematic way of designing the data flow aspects of a coordination scenario, orthogonally to the way in which the control flow is designed. Starting from a set of fundamental and basic building blocks, each data dependency will yield a data flow design that takes a set of design criteria (e.g. loose coupling, data confidentiality, etc.) into account. The pattern language is evaluated in three ways. First, it is shown that every potential coordination scenario for managing a data dependency can be composed by the set of patterns. Second, the pattern language was applied in a real-life insurance case to show how it can guide the design of complex data flows. Third, the patterns were implemented in a tool that provides configurable model-to-code transformations for automatically generating BPEL coordination scenarios. In this tool both the data flow and control flow can be designed separately using different sets of patterns.

Keywords: data dependencies, data flow, service composition, service coordination, coordination logic, patterns, guided design

1. Introduction

To stay competitive, businesses are forced to constantly optimise their performance and adapt flexibly and rapidly to the evolution of business goals and requirements. Therefore, many organisations transform from stable and monolithic enterprises into dynamic and distributed service-oriented enterprises. Similarly, the underlying information systems that support the organisations' business processes are more and more built by combining software services into loosely coupled, distributed, flexible, dynamic and adaptive service-based systems.

Service-based systems are mostly implemented using a Service Oriented Architecture (SOA) (Metzger and Pohl, 2009). A central idea in an SOA is the (hierarchical) composition of multiple services. Such service compositions are typically *process-based*, which means that a business process model serves as the blueprint for the service composition (Barros et al., 2005). Composing services into service-based systems requires the design of *coordination logic*, that describes all service interactions realising the composition.

Coordination as a research topic is not limited to computer science. It is also studied in disciplines such as organization theory, operations research, economics, linguistics, and psychology. Malone and Crowston (1994) created a more generic coordination theory, which defines coordination as the *management of dependencies between activities*. This definition is based on the intuitive idea that there is nothing to coordinate without any interdependence. In the

*Corresponding author
Email addresses: geert.monsieur@econ.kuleuven.be (Geert Monsieur), monique.snoeck@econ.kuleuven.be (Monique Snoeck), wilfried.lemahieu@econ.kuleuven.be (Wilfried Lemahieu)

context of process-based service composition, one can distinguish between two types of dependencies (Yang et al., 2002; Papazoglou et al., 1997). First, a *sequence dependency* between a service s_1 and a service s_2 indicates that the start or continuation of the execution of service s_2 depends on the completion of the execution of s_1. Second, a *data dependency* between a service s_1 and a service s_2 indicates that the start or the continuation of the execution of service s_2 depends on data that is provided by s_1. Papazoglou (2005) and Papazoglou and Van den Heuvel (2007) describe the *coordination* function that a composite service needs to perform as follows: *"controlling the execution of component services, and manage data flow among them and to the output of the component service (e.g. by specifying workflow processes and using a workflow engine for run-time control of service execution)"*. Controlling the execution of services ('control flow') and coordinating the 'data flow', is exactly what managing sequence and data dependencies is about. Note that in this article the term sequence dependency is used to refer to any kind of order dependency among activities in a process model (e.g. sequential, concurrent, etc.)

At present, coordination logic is mostly defined in an ad hoc way, based on the design knowledge and experience of the individual designer. Rigorous and systematic design guidelines on how to design appropriate coordination logic are still missing (Monsieur, 2010). Moreover, in languages for specifying coordination scenarios (e.g. BPEL (OASIS, 2007)) the control flow and data flow are typically intertwined, i.e. messages that represent a service request (control flow) also hold the input data for the service they trigger (data flow). In this way, the scenario that is chosen to manage the control flow is also imposed on the data flow. For example, a centralized control flow implies that the data flow is also centralized, with all data exchange passing through a single hub (e.g. the orchestration engine). The lack of a systematic approach and the intertwining of control flow and data flow considerations hampers the development of a systematic and automatable approach to service composition and hence hampers the evolution towards self-adaptive systems. We advocate to address the issues of managing sequence dependencies and data dependencies separately. Our previous research tackled the problem of managing sequence dependencies (Monsieur et al., 2010). This article presents a set of patterns that help to construct a *data flow* that manages a given set of data dependencies in a service composition in a systematic way, optimised according to a number of criteria. Examples of such criteria are data confidentiality, performance, loose coupling and robustness to change. Patterns are a means to capture and expose design knowledge that otherwise resides in the head of designers. As such, the use of this set of basic patterns facilitates the definition of the initial coordination logic at design time and at the same time, facilitates the adaptation process substantially as it provides a systematic way to build coordination protocols out of basic building blocks. Ultimately, the patterns (both for data dependencies and sequence dependencies) form the basis of configurable model-to-code transformations that will enable the (semi)automatic construction of coordination logic based on the evaluation of the criteria taken into account by each pattern.

The next section presents the research problem by means of the running example for this article. Subsequently, in Section 3 we describe the research methodology. Section 4 forms the core of this article and proposes a pattern language for managing data dependencies. In Section 5 this pattern language is evaluated. Related work is discussed in Section 6. The article ends with a conclusion in Section 7.

2. Running example and problem description

As a running example, we deliberately chose a non-automated example from the domain of hospital services so as not to clutter the discussion with implementation issues, but the approach and techniques presented in this article are equally applicable to software services. A service composition example with actual software services can be found in the insurance case study discussed in Section 5.2.2.

In a hospital nurses provide several (business) services to patients, such as taking care of patients with high fever. A business process that realizes this 'treating fever service' is shown in Figure 1 and could consist of the following tasks: check the patient's previous febrifuge usage in the medical records, obtain a febrifuge, give the febrifuge to the patient, measure the body temperature and register the body temperature.

The business process can be implemented by consuming four main services: the medical records service, the pharmacist's service, the doctor's service and the nurse's service. In this example we assume that the nurse plays the role of service composer. The medical records service must be consumed for retrieving information concerning previous usage of febrifuges, registering the febrifuge given and registering the current body temperature. To obtain a febrifuge the nurse should request this service from the pharmacist. Hence, the nurse can be considered as a service composer that needs to consume the service of a pharmacist. Both aspirin and paracetamol are fever reducers.

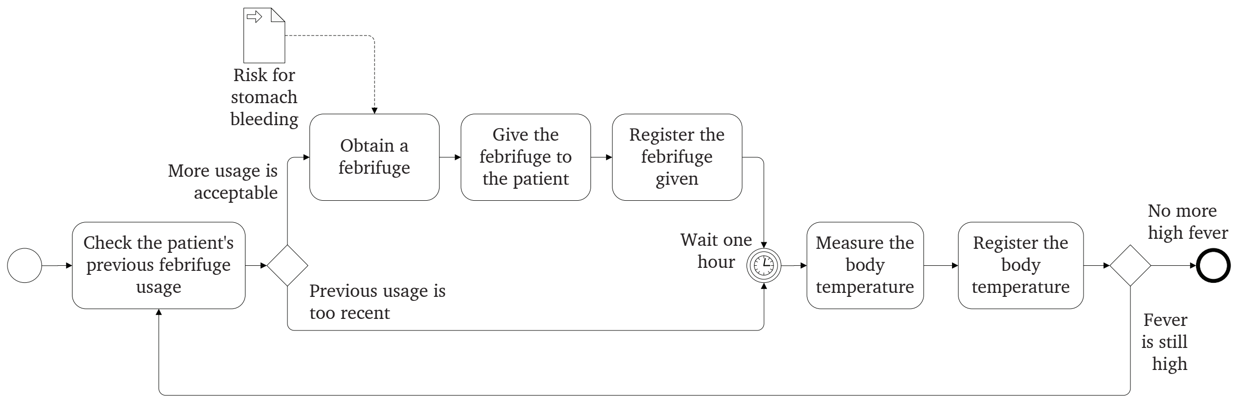

Risk for stomach bleeding
Obtain a febrifuge
Give the febrifuge to the patient
Register the febrifuge given
More usage is acceptable
Check the patient's previous febrifuge usage
Previous usage is too recent
Wait one hour
Measure the body temperature
Register the body temperature
No more high fever
Fever is still high

Figure 1: A business process for taking care of patients with high fever (represented using BPMN (OMG, 2010))

However, aspirin has the unpleasant side effect that it can cause stomach bleeding in certain circumstances. Therefore, the pharmacist needs information concerning the risk for stomach bleeding, before he or she can deliver an appropriate febrifuge. The risk for stomach bleeding is only known to a patient's doctor. This means that the doctor provides a second service that needs to be consumed in order to support the task of obtaining a febrifuge. We further assume that the nurse can complete the 'give the febrifuge to the patient' and 'measure the body temperature' tasks without the consumption of other services.

Even this rather simple service composition already demonstrates the need for coordination. For example, coordination is required to ensure that the registration of a febrifuge in a patient's medical record (consuming the medical records service) only occurs when a febrifuge is successfully given to the patient (consuming the nurse's service). Service coordination guarantees that sequence constraints as specified in the business process are met by constructing an appropriate control flow.

However, dealing with data needs is also part of service coordination. Services can require certain input data, which may in turn be the output from another service. Service coordination controls when which service is invoked, how input data is delivered and what to do with a service's output (Janssen and Feenstra, 2008) and as such defines an appropriate data flow. For example, the pharmacist needs the information concerning the risk for stomach bleeding, which is held by the doctor. Hence, there is a data dependency to manage between the pharmacist and the doctor: service interactions between the pharmacist and doctor must be coordinated, such that the pharmacist obtains the right information at the right time and in the right format.

Notice that that the structure of data objects and their interrelations, as part of an enterprise wide data model, have direct influence on the set of data dependencies that exist in service compositions. However, deriving all data dependencies from data models does not fall within the scope of this article. The main goal of this article is to present an approach that deals with the design of coordination scenarios, to manage a set of data dependencies, whereas the data (and sequence) dependencies themselves are considered as a given.

Even in this small example of data needs, many coordination scenarios are possible. In figures 2(a) and 2(b) two ways of managing the data dependency between pharmacist and doctor are shown. The question for the design of service systems is: what are possible scenarios and which one is the most appropriate in our situation? In this article we look for fundamental building blocks that allow constructing any possible scenario and that allow us to determine which coordination scenarios fulfill a component service's data needs in the most appropriate way. We therefore consider questions like: What are the fundamental differences between two coordination scenarios (e.g. figure 2(a) versus figure 2(b))? What are advantages and disadvantages of specific coordination scenarios?

3. Research question and methodology

Coordination logic for managing data dependencies realizes the data flow in a service composition. In the academic literature one can find several approaches that cater for alternative data flows. In section 6 we discuss these approaches and compare them to the one presented in this article. Most studies allow finding alternative data flows,

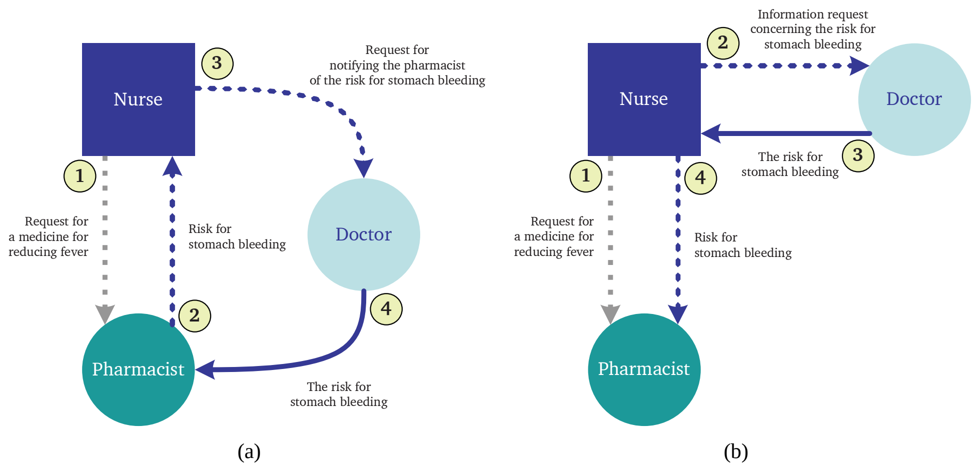

Nurse
Doctor
Pharmacist
1
Request for a medicine for reducing fever
2
Risk for stomach bleeding
3
Request for notifying the pharmacist of the risk for stomach bleeding
4
The risk for stomach bleeding
(a)
Nurse
Doctor
Pharmacist
1
Request for a medicine for reducing fever
2
Information request concerning the risk for stomach bleeding
3
The risk for stomach bleeding
4
Risk for stomach bleeding
(b)

Figure 2: Two ways of coordinating the pharmacist and doctor

but do not provide a *systematic* way of building different coordination styles nor do they analyze the advantages and disadvantages of alternatives. Because of the lack of a systematic way of discovering and assessing the alternatives, we cannot be sure such approach exhaustively identifies all potential scenarios to manage a set of data dependencies and does not overlook a potentially interesting scenario. Furthermore, as advantages and disadvantages are not discussed in depth, these approaches do not provide any help in choosing between alternative scenarios (if such alternatives are present at all).

If we define a *coordination scenario* as a specific set of service interactions constituting the coordination logic in a service composition, we can formulate the following research question:

Can we come up with a systematic way of composing coordination scenarios from fundamental building blocks so that the coordination scenario takes all data dependencies and specific design criteria (e.g. service coupling, data confidentiality, etc.) into account?

The building blocks proposed in this paper are patterns that together form a so called pattern language. In its simplest form a pattern can be defined as one or more *solution(s)* to a recurring *problem* that arises within a specific *context* (Buschmann et al., 2007). The design of coordination logic is a recurring problem in service composition, which explains our choice for patterns. Furthermore, as we will describe in Section 4.1 an SOA creates a context in which several forces are present. These forces can be used as evaluation criteria for constructing the most appropriate coordination scenario in a particular context. Each solution in a pattern balances these forces differently, which helps developers to choose between different patterns.

The process of discovering and documenting patterns is called pattern mining or pattern crafting. Usually pattern languages are developed by mining current practice using one of the following techniques: mining by interviewing, mining by borrowing, mining by teaching pattern writing, mining in workshops, mining your own experience (Rising, 1998, 1999). In this research we follow a slightly different and innovative approach. On the one hand, we mined current practice by looking at BPEL (OASIS, 2007) and WS-CDL (W3C, 2005) examples in academic and industrial literature (e.g. the service interaction patterns by Barros et al. (2005) or related work discussed in Section 6), running examples (e.g. in the BPEL standard (OASIS, 2007)) and generic scenarios identified in industry standards (e.g. RosettaNet Partner Interface Protocols (RosettaNet, n.d.)) (i.e. this could be labelled 'mining by borrowing'). Additionally, since we look for a set of patterns that should enable us to build any possible scenario, we looked for potential missing solutions (gaps in current practice) by elaborating the patterns from a theoretical perspective.

The search for adequate patterns can be classified as design-science research (Hevner et al., 2004). An important step in the Design Science Research Methodology (DSRM) (Peffers et al., 2007) is the definition of research objectives that can be tested to evaluate the artifacts that are proposed in this article. In pattern literature, patterns are mostly validated by presenting three known uses. We go beyond this rather weak form of validation. Next to defining the practical utility of the patterns as guidelines for developers, we also demonstrate the completeness of the pattern languages (i.e. no gaps due to incomplete (mining of) current practice) and the utility of patterns as basis for configurable model-to-code transformations. The following three testable research objectives describe what we expect from a pattern language for managing data dependencies:

- By combining the building blocks in several ways, it should be possible to construct every potential coordination scenario. In Section 5.1 we show how this objective is accomplished.
- The set of patterns should guide developers in choosing and combining the building blocks into a coordination scenario that takes specific design criteria into account. It is not our intention to focus on the criteria themselves or to provide an exhaustive list of design criteria. Instead we expect to come up with a generic framework that contains an initial set of design criteria and helps to include these criteria (or any set of criteria) in the design process of coordination scenarios. Section 5.2 discusses how we targeted this objective and demonstrates how design guidelines can be used to apply the patterns in an appropriate way.
- Based on the patterns it should be possible to semi-automate the construction of a coordination scenario by letting developers pick specific building blocks for dependency management and automatically generate an executable coordination from a business process specification. Section 5.3 presents a tool for pattern-based coordination of sequence and data dependencies. This implementation also demonstrates the practical utility of the set of patterns presented in this article, which is an important evaluation aspect in design science Hevner et al. (2004).

4. A pattern language for managing data dependencies

4.1. Introduction

As discussed below, the problem of managing a data dependency is divided into three subproblems, each addressed by a separate pattern. Each pattern describes all potential solutions to the subproblem and explains how each solution addresses particular forces (or not). Although each pattern deals with a specific recurring problem in data dependency management, the three patterns share a common context and the solutions described in a pattern are under influence of the same set of forces.

Common context

In the context of a service request involving a data dependency three relevant participants can be distinguished. We refer to an entity that consumes a particular service for requesting the execution of a business process task as a *Service Requester*. In a service composition this entity typically holds a (partial) description of the control flow, because it knows when to consume a specific service. The service that receives requests for executing a business process task is called a *Service Provider*. Possibly, a Service Provider needs certain data for processing the Service Requester's request. We refer to this kind of Service Provider as a *needy* Service Provider. The service that can provide the data needed is referred to as the *Data Provider*[1].

The previous definitions imply that a data dependency is always related to a Needy Service Provider and a Data Provider. In the context of the hospital example we can consider the nurse as a Service Requester that sends a request to the service provided by the pharmacist. Hence, the pharmacist plays the role of a Service Provider. Since the pharmacist needs information that is known by the doctor, the pharmacist can be considered as a Needy Service Provider and the doctor can be labeled as a Data Provider.

Obviously the distinct roles of *Service Requester*, *Service Provider* and *Data Provider* are relative. Depending on the interaction at hand, each partner in the interaction plays one of the roles or even combines several roles. A *Service Requester* requests the execution of a particular business process task. The *Service Provider* executes this task; such task can be 1) update a state (e.g. confirm an order or process a request for a febrifuge), 2) deliver data (e.g. the price of a product or the risk of stomach bleeding), or 3) call upon other services to execute a subtask. In the second case the service request is also a data request and hence the *Service Provider* is also a *Data Provider*. In the third case the

[1]In the pattern language we abstract from the fact that a Data Provider can be either the entity that owns the data or the entity that functions as a data mediator (Gamma et al., 1995) that can forward data requests to the right entities (e.g. other Data Providers or data owners).

Service Provider also becomes a *Service Requester*[2]. When designing data flows it is important to know that each of these kinds of tasks may need input data from a third party: the *Data Provider*. Obviously, the *Data Provider* can also be considered as a *Service Provider* as ”delivering data” can be considered as a service. However, in the context of this research, we wish to focus only on the perspective that the *Data Provider* should deliver data and that this data should attain the *needy Service Provider* somehow. The request for data may be issued (depending on the pattern) by the *Service Requester*, the *Service Provider* or even the *Data Provider*.

The three role names are stable in the sense that they do not vary in terms of the chosen data provisioning scenario. The Data Provider is always the Data Provider, independently of the scenario. The roles of Service Provider and Service Requester are also independent of the chosen data provisioning scenario since they refer to aspects of the *service request* independent of the data request. Note, however, that whereas the roles are stable within an interaction (i.e. a scenario in which a *Service Requester* sends a service request to request the execution of business process task), they are ’relative’ across different interactions: e.g. the Data Provider in one interaction may very well act as a Service Requester in another interaction (or in a subtask that was triggered in the context of the first interaction).

Finally one should note that it is also possible to merge the role of the data provider with the role of the service requester. In the further presentation of the patterns, we will however assume that the roles are performed by three different parties. Similarly, one can understand that merging the data provider with the service provider means that the service provider will have to consult its own data to deliver the service. In a similar way merging the service requester with the service provider would mean that the service requester requests a service from itself (while still needing data from a data provider).

Three subproblems

By analyzing similarities and differences between scenarios mined from current practices, we can conclude that a specific coordination scenario that manages a data dependency should answer three questions (see Figure 3):

1. Who decides that data is required from the Data Provider? (*data flow initiation*)
   (e.g. Who decides that information on the risk for stomach bleeding is required from the doctor?)
2. Who sends a request to the Data Provider? (*data request*)
   (e.g. Who sends the actual request for the risk for stomach bleeding to the doctor?)
3. How does the data flow between the Data Provider and the Needy Service Provider? (*data transmission*)
   (e.g. How does the risk information get from the doctor to the pharmacist?)

Different answers are possible for each of these questions. Per question, a pattern helps to answer the question in the most optimal way by taking advantages and disadvantages into account. Answers to the questions can then replace the cloud in figure 3. As such these three patterns form the three building blocks for scenarios that manage a data dependency between a Needy Service Provider and a Data Provider. While related work on alternative data flows in service compositions is mainly focused on optimizing the performance (i.e. reducing communication overhead, etc.) (see Section 6), the pattern-based approach proposed in this paper analyzes the advantages and disadvantages of each design alternative by considering *multiple* evaluation criteria or pattern forces, including robustness to change, loose coupling and data confidentiality. In each pattern different solutions to the same subproblem are described[3]. In each solution forces are balanced differently, resulting into different solution evaluations. By giving a weight to the evaluation criteria, service composers can be guided towards the most appropriate coordination scenario in the context at hand.

Common forces (evaluation criteria)

In contrast to related work the approach presented in this paper allows one to consider *multiple* advantages and disadvantages of each design alternative based on a set of evaluation criteria or pattern forces. In order to determine relevant forces, we studied the literature on service composition (e.g. (Balasooriya et al., 2005; Barros et al.,

[2]Of course, in practice, requests for these different kinds of tasks are sometimes combined into one single request, implying that the consumed service combines several roles.

[3]The pattern language we developed for managing sequence dependencies contains different patterns that all describe one solution to one and the same problem. In contrast, the pattern language for managing data dependencies consists of three patterns, each addressing one specific problem. Furthermore, each pattern describes different solutions to one (pattern-specific) problem. In the literature one can find both kinds of patterns (Paikens and Arnicans, 2008).

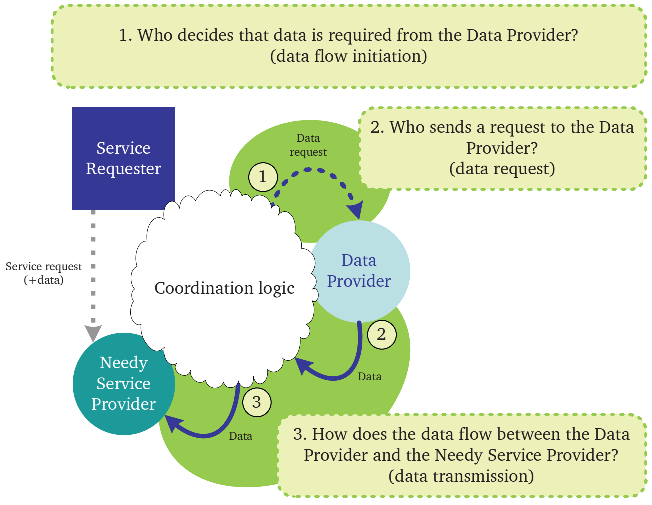

1. Who decides that data is required from the Data Provider? (data flow initiation)
Service Requester
Data request
2. Who sends a request to the Data Provider? (data request)
1
Service request (+data)
Coordination logic
Data Provider
2
Data
Needy Service Provider
3
Data
3. How does the data flow between the Data Provider and the Needy Service Provider? (data transmission)

Figure 3: Three questions that need to be answered by a specific coordination scenario that manages a data dependency

2005; Erl, 2007; Goethals, 2008; Haesen et al., 2006; Zirpins et al., 2004; Legner and Vogel, 2007; Paci et al., 2008; Habala et al., 2008)). This literature review yielded the eight forces mentioned below. This set of forces is complete as far as literature is considered. Practical experience may reveal other criteria in the future. This is not a fundamental problem as the same approach could be applied with a completely different set of forces. We discuss the forces briefly below. More details about them, including references to the literature, can be found in Appendix A.

EC1 *Robustness to change*: In a service-oriented environment it is critical that the propagation of changes due to the modification of the interface of a service is minimized. Therefore, a change in the data requirements should have minimal impact on the way in which the Service Provider is consumed.

EC2 *Adjustability*: This criterion is about the ability to change which data is sent to the Service Provider in function of a specific service request. For example, when managing the data dependency between the pharmacist and the doctor, it could be that only information regarding the patient at hand is sent to the pharmacist or it could be that each time the same fixed set of patients and patient information is sent.

EC3 *Coupling with Data Provider*: In some situations the data needed is not always provided by the same Data Provider. Each time there is a change of Data Provider the party that is sending data requests to the Data Provider needs to be notified and modified properly. Similarly, if the new Data Provider has a different interface, a change in the implementation of the party that is interacting with the Data Provider is required.

EC4 *Data provider accessibility*: Sometimes it is possible that the Service Provider or Service Requester does not know *which* Data Provider can provide the required data (e.g. the pharmacist does not know who is the patient's doctor). In other cases there can be access restrictions.

EC5 *Confidentiality of data requirements*: It can occur that a Service Provider's data requirements are confidential (e.g. suppose that nurses cannot have insight into the pharmacist's internal decision processes), which means that only a limited set of services or even only the Service Provider itself knows which data is needed in the process of delivering its service.

EC6 *Data confidentiality*: When requesting a Data Provider to send the required data to an entity, it is important to realize that the provided data can be confidential and therefore there can exist a need to limit the number of entities that the Data Provider can share the data with.

EC7 *Data reusability*: In some business cases data provided by a Data Provider is used by more than one Service Provider. In such situations an optimal coordination scenario limits the number of data requests that are sent to the Data Provider.

EC8 *Data format*: When the Data Provider replies, the data provided is possibly not in a form that is expected by the Service Provider. For example, the data format needs to be adapted, or the data should be made anonymous. In short, in some cases data transformations are desirable before the data is received by the Service Provider. Dealing with different data formats is a common challenge when information is shared among services.

As discussed in the related work section, there exist many research papers focusing on optimizing the performance of data flows, which mainly deal with the centralization versus decentralization question and the resulting number of

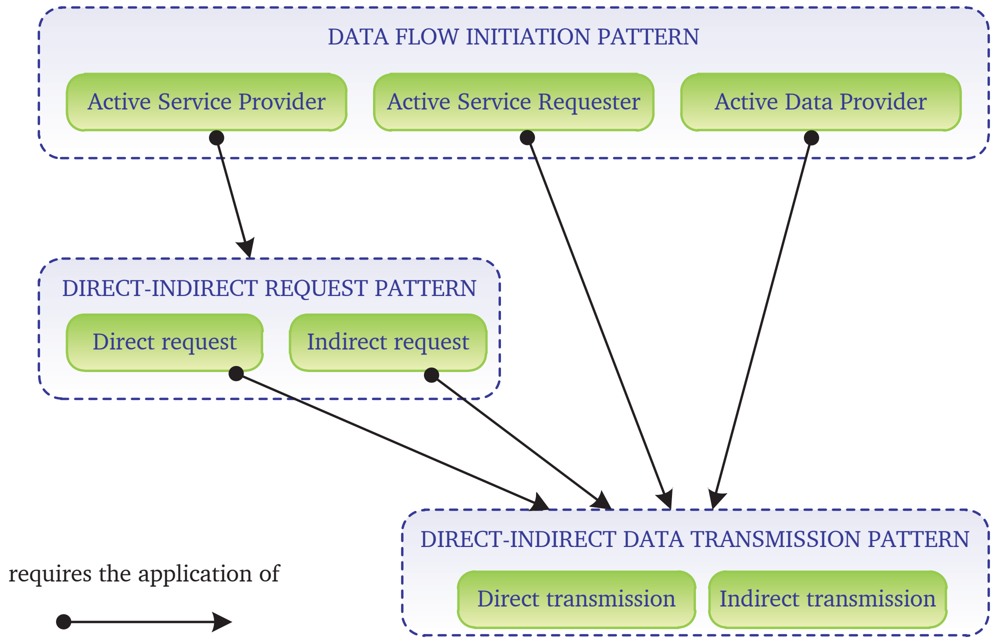


Figure 4: Relationships between the three patterns

message exchanges. However, although performance certainly is an important force for data flow design, we feel it is not a pure conceptual design issue, as dealing with performance requires a lot of knowledge about which implementation technology will be used. It is for example easy to determine the number of messages sent in a coordination scenario, but this does not necessarily have to correlate with the performance. Indeed, if the implementation technology used to realize the data flow entails a lot of overhead for each message exchange, however big or small the message is, it would be favorable to opt for a scenario with as few message exchanges as possible, even if this results in some data being sent that is actually not needed by the Service Provider (as in the Active Data Provider scenario, cf. below). However, if it is mainly the size of the data transfers (rather than the number) that impacts performance, the respective scenarios would be evaluated completely differently against this force. Other implementation related factors may have an impact as well. Therefore, we decided not to include this force in the evaluations. However, once an implementation technology has been decided upon, the proposed pattern language is perfectly fit to evaluate the different scenarios, as implemented through this technology, against the performance forces.

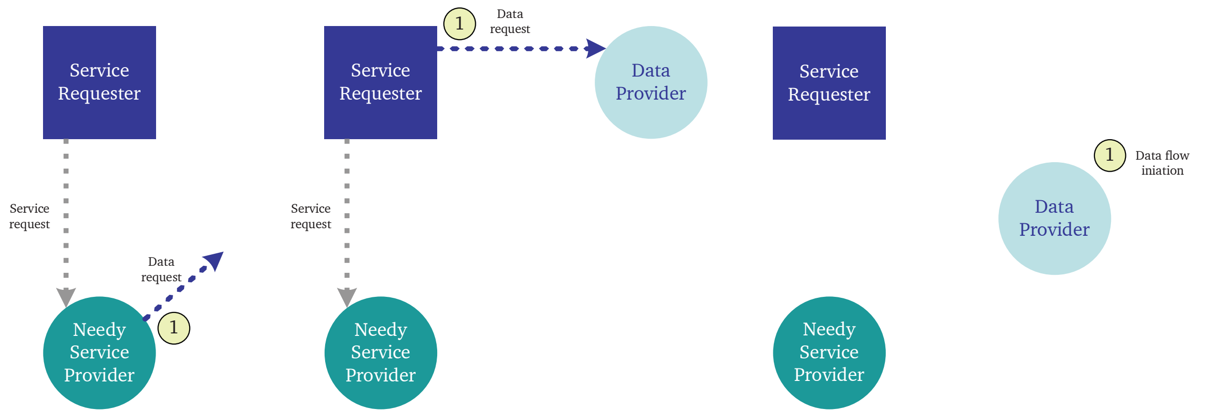
4.2. Pattern overview

Three patterns are used to manage the three aspects of a data dependency: DATA FLOW INITIATION (see Subsection 4.3), DIRECT-INDIRECT REQUEST (see Subsection 4.4) and DIRECT-INDIRECT DATA TRANSMISSION (see Subsection 4.5). Each pattern consists of several solutions, among which a service composer can choose by considering the evaluation criteria and the solutions' consequences. In Figure 4 each pattern is visualized in a box (dashed line border) containing both the pattern name and sub-boxes referring to different solutions in that pattern. The relationships between the three patterns are indicated by arrows. An application of the DATA FLOW INITIATION pattern can function as a first necessary step in managing data dependencies. The next steps toward a coordination scenario is indicated by means of the arrows. An arrow pointing from a pattern sub-box A to a pattern box B indicates that the pattern represented by B should be applied next when a pattern is applied in the way represented by sub-box A. Although theoretically the three patterns can be applied in any order, the order shown in Figure 4 is the most intuitive one and will therefore be used further on.

4.3. Data flow initiation pattern

4.3.1. Problem

If a Service Requester sends a request to a Service Provider, it can occur that the Service Provider does not possess sufficient data for completing its internal processing. Therefore additional input data should be collected from other



(a) Active Service Provider (b) Active Service Requester (c) Active Data Provider

Figure 5: Data flow initiation

services, which requires a data flow. For example, when a nurse asks the pharmacist a febrifuge for a certain patient, the pharmacist may need more input data (e.g. the risk for stomach bleeding). This raises an important question regarding the data collection process: **Who decides that data is required from a Data Provider?**

4.3.2. Solutions

There are three possible data flow initiators in a coordination scenario (see Figures 5(a),5(b), and 5(c)). First, an active Service Provider initiates the data flow by sending out a data request (see step one in Figure 5(a)). It is not specified to which entity the Service Provider sends the request; this is discussed in the direct-indirect request pattern described in Subsection 4.4. Second, an active Service Requester initiates the data flow by sending a data request to the Data Provider (see step one in Figure 5(b)). Third, while in the previous two scenarios the Data Provider sends out data upon request of another entity (i.e. the Service Provider or Service Requester respectively), in a scenario with an active Data Provider it is the Data Provider itself that decides that it needs to send data (see Figure 5(c)).

4.3.3. Consequences (evaluation of the solutions)

The solutions presented in Section 4.3.2 should be evaluated against each force discussed in Section 4.1. Table 1 summarizes all consequences of the data flow initiation pattern without prioritizing any force or consequence. When applying the pattern in practice the context at hand will have to be considered to determine the most relevant forces to retain for developing the solution for the case at hand.

For detailed discussion on these consequences we refer to Appendix B.1.

4.3.4. Relationship with other patterns

An active Service Provider sends out data requests in order to receive the missing input data (see step one in Figure 5(a)). The direct-indirect request pattern shows who contacts the actual Data Provider with the request (see Subsection 4.4). An active Service Requester sends data requests to the Data Provider (see step one in figure 5(b)). As a consequence the Data Provider sends out data. In case of an active Data Provider the Data Provider itself decides if it needs to send out data. In each of these cases, the direct-indirect transmission pattern shows how the data flows from the Data Provider to the Needy Service Provider (see Subsection 4.5).

4.4. Direct-Indirect request pattern

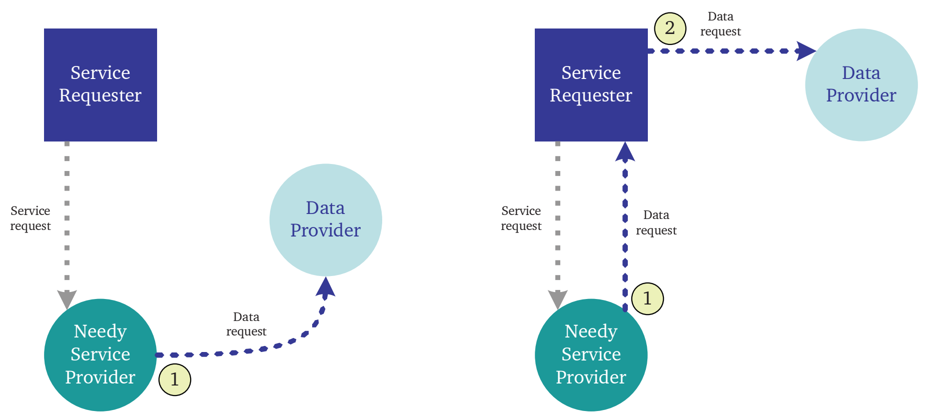
4.4.1. Problem

If a Service Provider is active, then the Service Provider sends out data requests in order to receive missing input data. This raises the following question: **Where can an active Service Provider send its data requests to?** For example, if a pharmacist wants to inform himself about the risk for stomach bleeding, the pharmacist needs to know who he can ask this question to. Should he ask the nurse or can he ask the doctor?

	Active SR	Active SP	Active DP
Robustness to change	-	+	-
Adjustability	+	+	-
Coupling with Data Provider	SR coupled	SP or SR coupled	no coupling
Data provider accessibility	SR needs access	SP or SR needs access	no access required
Confidentiality of data requirements	-	*depends on request*	*depends on data transmission*
Data confidentiality		*depends on data transmission*	
Data reusability		*depends on data transmission*	
Data format		*depends on data transmission*	

SR = Service Requester
SP = Service Provider
DP = Data Provider

Table 1: Summary of the consequences of data flow initiation



(a) Direct request (b) Indirect request

Figure 6: Direct request versus indirect request

4.4.2. Solutions

An active Service Provider can send its data requests to two entities, as shown in Figures 6(a) and 6(b). First, an active Service Provider can send a *direct request*, which means that the data request is sent directly to the Data Provider. Second, an active Service Provider can send its data request to the Service Requester (see step two in Figure 6(b)), which is supposed to forward the data request to the appropriate Data Provider (see step three in figure 6(b)). This alternative is referred as an *indirect request*.

4.4.3. Consequences (evaluation of the solutions)

The solutions presented in Section 4.4.2 should be evaluated against each force discussed in Section 4.1. Table 2 summarizes all consequences of the direct-indirect pattern without prioritizing any force or consequence. When applying the pattern in practice the context at hand will have to be considered to determine the most relevant forces to retain for developing the solution for the case at hand. For a detailed discussion on these consequences we refer to Appendix B.2.

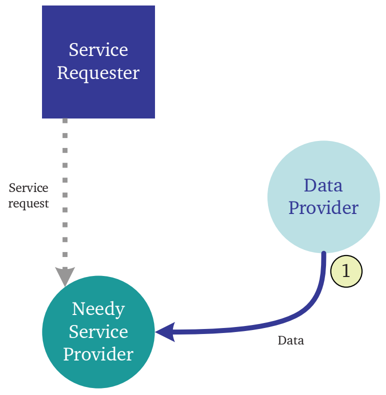
4.4.4. Relationship with other patterns

In both scenarios the Data Provider receives a data request (see step one in Figure 6(a) and step two in Figure 6(b)). As a consequence, data should be received by the Service Provider. The direct-indirect transmission pattern shows how the data flows from the Data Provider to the Needy Service Provider (see Subsection 4.5).

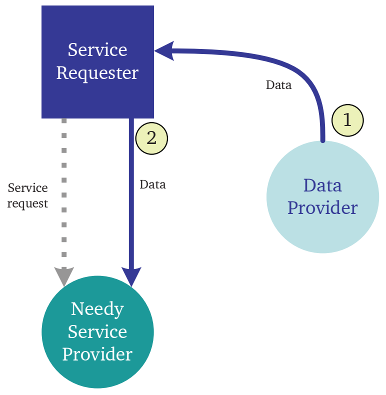
	Direct request	Indirect request
Robustness to change	(+)	(+)
Adjustability	(+)	(+)
Coupling with Data Provider	SP coupled	SR coupled
Data provider accessibility	SP needs access	SR needs access
Confidentiality of data requirements	+	-
Data confidentiality	*depend on data transmission*	
Data reusability	*depend on data transmission*	
Data format	*depend on data transmission*	

SR = Service Requester
SP = Service Provider
DP = Data Provider
(+) is inherited from Active SP

Table 2: Summary of the consequences of DIRECT-INDIRECT REQUEST



(a) DIRECT DATA TRANSMISSION



(b) INDIRECT DATA TRANSMISSION

Figure 7: DIRECT DATA TRANSMISSION versus INDIRECT DATA TRANSMISSION

4.5. Direct-Indirect transmission pattern

4.5.1. Problem

If a Data Provider received a data request, the requested data should be delivered to the Service Provider. If an ACTIVE DATA PROVIDER is used in a coordination scenario, the Service Provider should also receive data sent by the Data Provider. In both cases, the following question comes up: **How does the data flow from the Data Provider to the Service Provider?**

4.5.2. Solutions

Data can flow from the Data Provider to the Service Provider in two ways. First, the Data Provider can initiate a *direct data transmission*, which means that the data is sent directly to the Service Provider (see Figure 7(a)). Second, the data can be transmitted from the Data Provider to the Service Requester and subsequently to the Service Provider. This alternative is referred to as an *indirect data transmission* (see Figure 7(b)).

4.5.3. Consequences (evaluation of the solutions)

The solutions presented in the previous section should be evaluated against each force discussed in Section 4.1. Table 3 summarizes all consequences of the DIRECT-INDIRECT DATA TRANSMISSION pattern without prioritizing any force or consequence. When applying the pattern in practice the context at hand will have to be considered to determine the most relevant forces to retain for developing the solution for the case at hand. One can see for example that issues with confidentiality will favor a direct transmission. Issues on the data format will on the other hand favor indirect transmission. As a result, in case both transformation and confidentiality are issues (or neither of them), data reusability may be the discriminating factor to choose between the alternatives. For a detailed discussion on these consequences we refer to Appendix B.3.

	Direct transmission	Indirect transmission
Robustness to change	*depend on initiation/request*	
Adjustability	*depend on initiation/request*	
Coupling with Data Provider	*depend on initiation/request*	
Data provider accessibility	*depend on initiation/request*	
Confidentiality of data requirements	*depends on initiation/request*	-
Data confidentiality	+	-
Data reusability	-	+
Data format	-	+

SR = Service Requester
SP = Service Provider
DP = Data Provider

Table 3: Summary of the consequences of direct-indirect data transmission

4.6. Combining the patterns into coordination scenarios

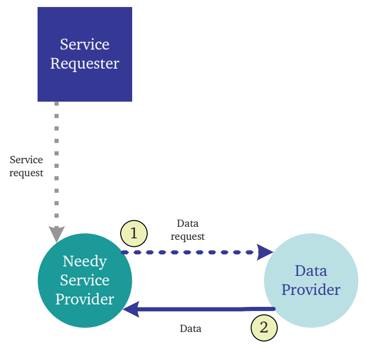
As described in the introduction to the pattern language (see Subsection 4.1) the three patterns discussed above are building blocks that need to be combined to build coordination scenarios that manage data dependencies.

Since the data flow initiation pattern has three solutions, and the direct-indirect request pattern and direct-indirect transmission pattern have two solutions, it is, in theory, possible to combine the patterns in twelve different ways. However, by following the pattern relationships that were discussed in Subsection 4.2 and shown in figure 4 only eight combinations are possible. This makes sense because, by definition, only in a coordination scenario with an active Service Provider it is relevant to decide whether the data request should be sent in a direct or indirect way. An active Service Requester sends data requests to the Data Provider in a direct manner. Active Data Providers do not receive data requests. Figures 8(a) to 8(h) represent the eight possible combinations[4]. Capitalized words in the figures' captions indicate which patterns have been applied.

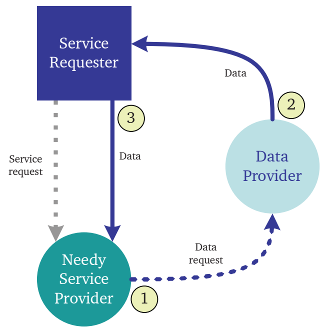
5. Evaluation

Section 3 defined three testable research objectives. In this section we evaluate how the pattern language achieves these objectives. First, Subsection 5.1 demonstrates that the theoretical elaboration of the pattern language indeed led to completeness of the language, meaning that it is possible to compose every potential coordination scenario using the patterns that constitute the language. Subsequently, Subsection 5.2 demonstrates the practical utility by describing how the pattern language can be used to guide the design of coordination logic. Finally, Subsection 5.3 demonstrates the practical utility of the patterns in a model-driven engineering approach: it describes how the pattern language was implemented in a tool that supports and automates the generation of BPEL coordination logic.

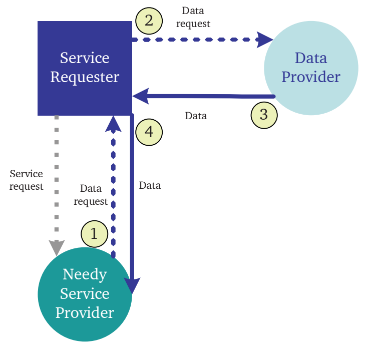
[4]As described in the introduction of this section (see 4.1) the roles of *Service Requester*, *Service Provider* and *Data Provider* are stable in these different scenarios. It is always the *Service Requester* that sends a *service request* to the *Service Provider*, while the *data request* (depending on the scenario) is sent by the *Service Requester*, *Service Provider* or (implicitly) by the *Data Provider* (i.e. no arrow/message representing a *data request* is drawn, meaning that the *Data Provider* itself triggers the data flow.)



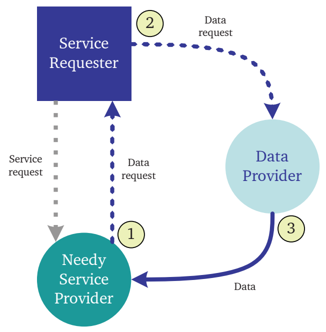
(a) Active SP with direct request and direct data transmission



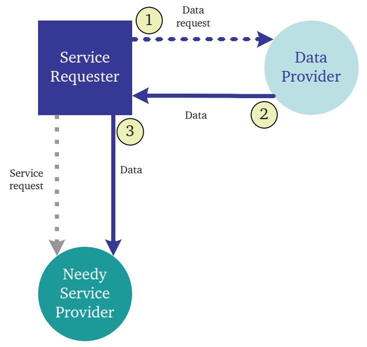
(b) Active SP with direct request and indirect data transmission



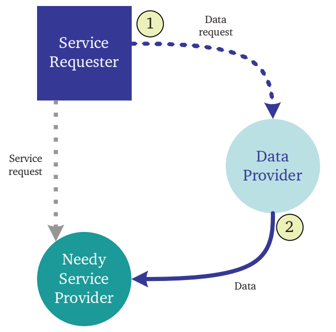
(c) Active SP with indirect request and indirect data transmission



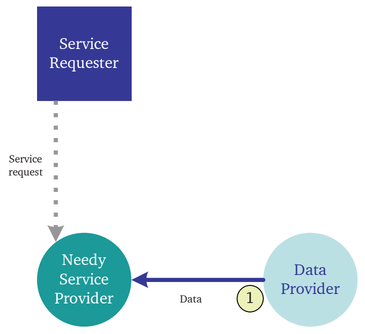
(d) Active SP with indirect request and direct data transmission



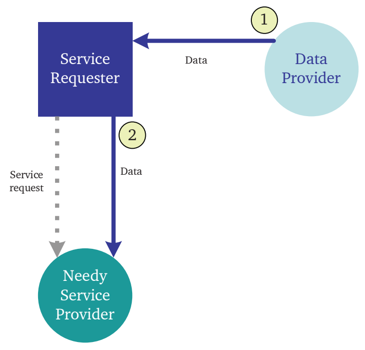
(e) Active SR with indirect data transmission



(f) Active SR with direct data transmission



(g) Active DP with direct data transmission



(h) Active DP with indirect data transmission

Figure 8: Eight possible combinations (SP = Service Provider, SR = Service Requester, DP = Data Provider)

5.1. Completeness confirmation

The patterns are innovative in the sense that they represent basic building blocks that can be combined to compose executable 'coordination scenarios'. A coordination scenario shows the actions taken by all involved parties to get the data from the Data Provider to the Needy Service Provider. Hence, the real value of the patterns depends on an evaluation of the *composition* of the patterns into concrete coordination scenarios. In this section we show that *all* potential coordination scenarios can be composed by combining the patterns (cfr. first research objective). In order to determine the set of all possible coordination scenarios we obviously cannot start from the three questions that form the basis for the three patterns (see Subsection 4.1). In order to independently calculate the universe of coordination scenarios, we declaratively specified what a coordination scenario should accomplish and in which message exchanges a Service Requester, Service Provider and Data Provider can be involved. For example, it is easy to understand that in every coordination scenario there needs to flow data from the Data Provider to the Service Provider.

We have used Prolog (Clocksin and Mellish, 2003; Wielemaker, 2003), a general purpose logic programming language, to calculate the complete universe of coordination scenarios. This declarative language has its roots in formal logic. Typically, a Prolog program logic is expressed in terms of relations, represented as facts and rules. A computation is initiated by running a query over these relations. This allows us to declaratively specify what a coordination scenario should accomplish (e.g. the service provider must receive data from an entity; the data provider must send data to an entity; etc.), so that an execution of the Prolog program (i.e. a query that calculates or derives all solutions) results into all possible coordination scenarios. The coordination scenarios found by the Prolog predicate exactly match the coordination scenarios that can be composed by combining the patterns, which confirms the completeness of the pattern language (cfr. first research objective).

Appendix C contains the complete Prolog program.

5.2. Guided design

In this subsection we describe how the pattern language can be used to guide the design of coordination logic. More specifically, we show how the pattern language helps to construct the most appropriate coordination scenarios in the running example and a real-life insurance case.

5.2.1. Running example

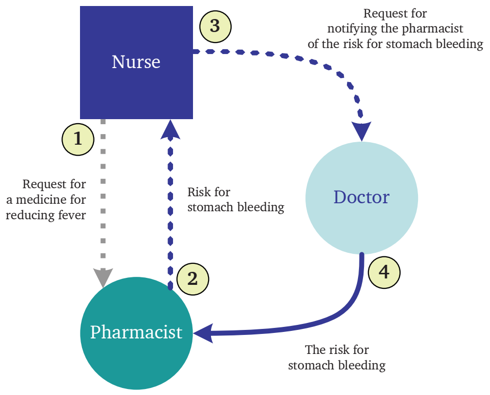
By following the pattern relationships as described in 4.2 and shown in figure 4 we can construct an appropriate coordination scenario for the hospital example (see Section 2 for problem description):

- Data flow initiation: Since neither nurses nor doctors want to keep track of which input data is required by the pharmacist, it is probably more desirable to choose an active pharmacist. Nurses simply want to use some services provided by the pharmacist. Furthermore, it is not preferred that changes in data requirements result in changes to how the nurses work (or consume the pharmacist's services).

- Direct-indirect request: This pattern needs to be applied, because pharmacists are considered as active service providers. Since the pharmacist does not know which doctor is treating the patient, it is preferred that the pharmacist asks the nurse for more information concerning the risk for stomach bleeding (see step two in figure 9). Subsequently, the nurse can forward the request to the right doctor (see step three in figure 9). Hence, an indirect request is the most appropriate choice.

- Direct-indirect data transmission: Suppose the risk for stomach bleeding is quite confidential information that can not be shared with the nurse. Then, the direct data transmission scenario is the best solution. Hence, the doctor should send the information concerning the risk for stomach bleeding directly to the pharmacist (see step four in figure 9).

The complete solution for this example is shown in figure 9.



Nurse
3
Request for notifying the pharmacist of the risk for stomach bleeding
1
Request for a medicine for reducing fever
Risk for stomach bleeding
Doctor
2
4
Pharmacist
The risk for stomach bleeding

Figure 9: Active pharmacist with indirect request and direct data transmission

5.2.2. *Insurance case study*

We validated the patterns by means of a real-life business case at a Belgian banking & insurance company (Haesen et al., 2006). As the case involves multiple data providers and a large set of data to which different forces apply in different ways, the case goes beyond the construction of a single coordination scenario: it will require multiple applications of the pattern language to different subsets of data. The case is therefore suited to demonstrate the practical utility in complex real life situations. The case can be considered a situation in which a consumer wants his house to be insured together with the house content. A simplified version of the business process consists of the following tasks: processing the customer's request, presenting an offer to the customer, making the contract, sending the insurance policy to the customer, and payment by the customer. In the context of this article we only consider the first task, which deals with the processing of customer requests. The system supporting this task, which we refer to as the *insurance request management (IRM) service*, is composed of several (component) services: insurance quote service, sales service, customer information service, blacklist service and external information service. The main service that is consumed for this task is the *insurance quote service*. This service accepts or rejects the request and needs to calculate the insurance premium in case of acceptance. This yields a two-staged approach. The acceptance step investigates whether or not to accept the request for insurance. The second step is the tarification step which generates a price offer. The first step requires a substantial amount of data in order to evaluate all possible reasons for rejection. On the other hand, a minimum of data may be sufficient to provide the customer with a first, rough estimation of the price, purely for informational purposes. These data requirements explain why, besides the insurance quote service, several other services are involved when composing the IRM service. The insurance quote service needs to be combined with other services because of specific (data) needs:

a **Information about existing customers**: The person who wants to have an object insured can be an existing or a new customer of the insurance company. For an existing customer most data will be available at the *customer information service*.

b **Data concerning descriptions of expensive items**: The premium of the house content depends on the fact whether the customer possesses exclusive and expensive goods, such as jewelry or special stamp collections. The premium increases proportionally to the value of those possessions. The data used to calculate the premium for the house content can be altered after the construction of the application. For example the premium for a stamp collection may initially only depend on the number of stamps in the collection. After examining past insurance claims, the insurance company may wish to consider also the exact kind of stamps for the premium calculation. This means further communication with the customer or interactions with an *external information service* containing price information of expensive objects, are needed.

c **Information about new customers**: All data about a new customer will have to be retrieved by the *sales service*, which interactively questions the customer.

d **Information about blacklisted customers or fraudulent family members**: Before the insurance request is accepted, the insurance quote service needs information about possible fraudulent family members. Furthermore, the insurance quote service needs to check whether the customer is present on any blacklists of untrusted payers. This information can be retrieved from a third party service which is referred to as the blacklist service in the rest of this section.

e **Base insurance quote**: Simplified, an insurance quote can be determined using a base insurance quote which is raised or reduced depending on the specific risk estimations. These risk estimations and the effect on the quote are calculated by the insurance quote service. The base insurance quote, however, is set by the sales service. Hence, in order to calculate the complete insurance quote, the base insurance quote must be retrieved from the sales service.

In summary, we can consider five main data needs, which all require some interaction with a service that needs to be included in the service composition. In terms of the terminology as used in this article, the insurance quote service plays the role of a Needy Service Provider, while the other services (e.g. sales service or third party service to check blacklists) play the role of the Data Providers. The entity that composes the IRM service is considered as the Service Requester, since it requests the insurance quote service. For each of the data needs, we follow the process described in Figure 4.1. For each step, we evaluate the criteria as described in see Subsections 4.3.3, 4.4.3 and 4.5.3, which leads to the selection of the optimal solution. Finally, for each data need the resulting coordination scenario is linked to one of the eight combinations that were discussed in Subsection 4.6.

a **Information about existing customers**: Information about existing customers can be retrieved from the customer information service of the insurance company. In order to choose the way to initiate the data flow, we need to consider criteria EC1-EC4 (see DATA FLOW INITIATION pattern in Subsection 4.3.3). We are in a situation of rather stable data, so robustness to change (EC1) is not so much of an issue. In terms of adjustability (EC2), we consider that the data is strongly related to the specific insurance quote request and that we therefore will prefer a solution that favors adjustability. Furthermore, we prefer loose coupling (EC3) with the data provider, that is to say between the insurance quote service and the customer information service. As a result of this evaluation of criteria EC1-EC4, we conclude that ACTIVE INSURANCE QUOTE SERVICE REQUESTER provides the best solution of data flow initiation. This means the insurance quote service simply expects to receive this information, which is acceptable because it is rather stable data and consumers do not have to worry that the interface and required data are changing frequently. The loose coupling between the insurance quote service and other services is guaranteed, because the Service Requester, which is responsible for triggering the insurance quote service, should send out a request for customer data to the customer information service. In the case of an ACTIVE SERVICE REQUESTER the pattern of DIRECT-INDIRECT REQUEST can be skipped. The next step is thus to determine the best way of data transmission. Two alternatives remain possible. Here we need to additionally consider criterion EC6-8 about the confidentionality of data requirements. If the customer information needed is evaluated as confidential, then DIRECT DATA TRANSMISSION is better than indirect data transmission (see EC6 evaluation in Subsection 4.5.3). However, in case data format (EC8) is an issue (e.g. when the customer information service does not provide the data in the correct form, transformation by the Service Requester is needed), the evaluation of this criterion justifies INDIRECT DATA TRANSMISSION via the Service Requester. Assuming that data reusabilityis not an issue and that in addition data confidentiality is important, the combined application of the patterns results in the coordination scenario represented in Figure 10(a) (based on direct data transmission).

b **Data concerning descriptions of expensive items**: For some customers additional data might be needed. For example, customers that have large collections of stamps need to be treated in a different way. A precise estimate of the value of the collection is needed to calculate the insurance premium. Therefore, an insurance quote service requires detailed descriptions of the stamps. Since this information is only needed in certain cases - only when the customer has exceptionally expensive items in his house we need to favor a solution that is robust to changes (EC1). For data flow initiation, this leads us to opting for an ACTIVE PROVIDER. When using an ACTIVE PROVIDER, the second step is to evaluate whether one needs a direct or indirect request. Since the insurance quote service does not have any knowledge on where to get this information, an INDIRECT REQUEST is necessary (see EC4, data provider accessibility). Next, as with the other types of data above, for this data two sorts of data transmission are possible too. First, in the case of confidential information (EC6) (e.g. the value of the items is extremely high), it is better to choose DIRECT DATA TRANSMISSION. Second, when transformation of the information (EC8) is a priority above confidentiality, INDIRECT DATA TRANSMISSION via the Service Requester is more appropriate. Assuming that data about the expensive items may need to be converted and that confidentiality is not a primary issue, the application of these patterns results in the coordination scenarios represented in Figure 10(b) (based on indirect data transmission).

c **Information about new customers**: For the information about new customers the patterns are applied in a similar way. The main difference is the data provider, which can be either a sales service or perhaps the Service Requester itself (e.g. collected when coordinating a previous process task). In any way, the insurance quote service prefers to have a loose coupling with this Data Provider (EC3), which (again) motivates the choice for an active insurance quote Service Requester for data flow initiation. Similar to the previous data need, both direct data transmission and indirect data transmission can be useful in certain situations. Assuming (again) that data reusability is not an issue and that in addition data confidentiality is important, the combined application of the patterns results in the coordination scenario represented in Figure 10(c) (based on direct data transmission).

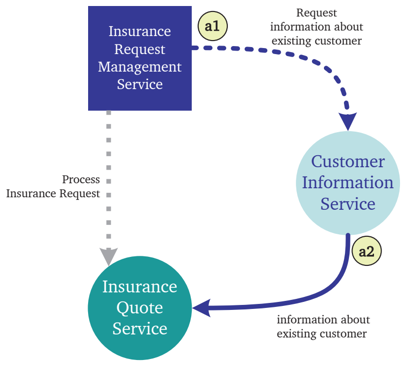
d **Information about blacklisted customers or fraudulent family members**: Checking whether or not a customer (or a family member) is on any blacklist, is only a thing that the insurance quote service can do, because only this service knows where to get this information. Furthermore, only the insurance quote service has access to the blacklist service. The evaluation of EC4, data provider accessibility, hence motivates the use of an active service provider (insurance quote service), using direct requests. Since this data is highly confidential (EC6), direct data transmission is also more appropriate in this case. These choices also support the fact that the insurance quote service prefers not to share its business rules (checking blacklists and/or family members) with its consumers (i.e. a Service Requester) (see EC5, confidentiality of data requirements, in 4.3.3). The application of these patterns results in the coordination scenario represented in Figure 10(d).

e **Base insurance quote**: For each type of insurance (car, house, etc.) there exists a base insurance quote, that is set by the sales service. Both the insurance quote service and the insurance quote Service Requester prefer a loose coupling (EC3) with the sales service who acts as a data provider in this case. Therefore, it is better to choose an active data provider (sales service) that sends a set of base insurance quotes to the insurance quote service from time to time (e.g. each time the sales service decides to modify base insurance quotes). In case of an active data provider no choice needs to be made about direct or indirect requests. Based on potential data transformation requirements (EC8), one can make a choice between direct data transmission and indirect data transmission. Assuming that no data transformation is required, the application of these patterns results in the coordination scenarios represented in Figure 10(e) (based on direct data transmission).

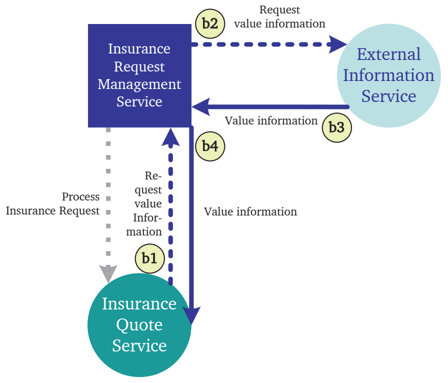
Hence, a final solution for the management of the data dependencies is constructed by combining several coordination scenarios, each taking care of a particular set of data (see Figure 11). As explained above, the insurance quote Service Requester takes the role of an active Service Requester with respect to customer data, while the insurance quote service takes the role of an active Service Provider when it comes to information about the insured items or confidential background data about customers. In the next subsection (see 5.3) we present a tool for pattern-based coordination and show how this tool uses the patterns to generate an executable coordination scenario.

The conclusion of this validation exercise demonstrates that at least seven out of eight combinations prove to be useful in practice. Moreover, it also demonstrates that there is no 'one size fits all' solution. The ideal solution can only be obtained by considering the specific characteristics of data and applying the suitable pattern for each different set of data. This allows one to balance the different requirements and meet several criteria at once. The solution developed by applying the patterns has been implemented at the banking & insurance company as a new version in replacement of the existing version because of its improved stability (robustness to change), its capability of handling confidential data and its satisfying performance level. Furthermore, when data requirements are changing at the company, the use of the patterns, including the guiding criteria, makes it easier to adapt the coordination scenarios than before when coordination logic was designed in an ad-hoc fashion. However, more research is needed to quantitatively evaluate that the use of the patterns contributes to a more efficient and effective development of coordination scenarios.

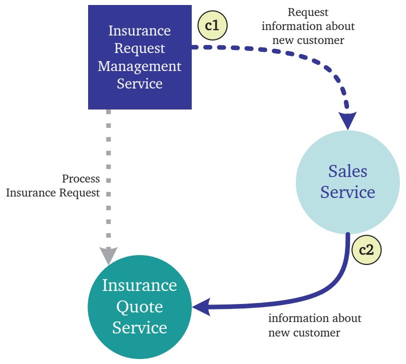
One combination of patterns, namely the scenario represented in Figure 8(b), has not been used in this real life case. Nevertheless, this does not imply that this pattern is useless. Since it is the result of a logical deduction step on the possible combination of the three basic patterns (active Service Provider with direct request and indirect data transmission), it has its place in the overview of potential solutions and might still prove useful in future real life cases.


Insurance Request Management Service
a1
Request information about existing customer
Customer Information Service
Process Insurance Request
a2
Insurance Quote Service
information about existing customer

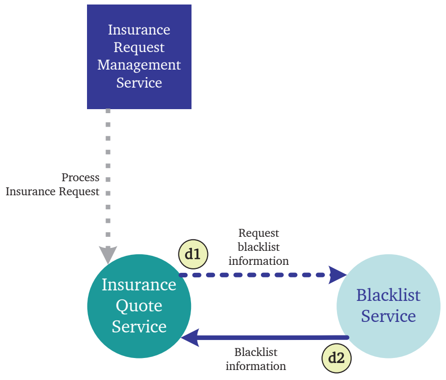
(a) Active SR with direct request (existing customer information)


Insurance Request Management Service
b2
Request value information
External Information Service
Value information
b3
b4
Process Insurance Request
Re-quest value Infor-mation
Value information
b1
Insurance Quote Service

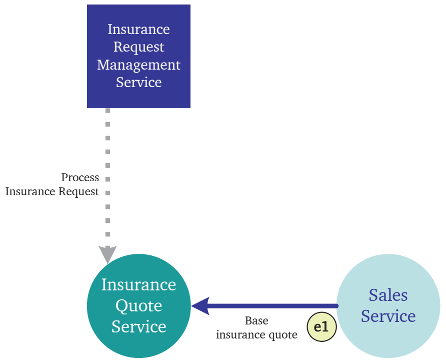
(b) Active SP with indirect request and indirect data transmission (value information


Insurance Request Management Service
c1
Request information about new customer
Sales Service
Process Insurance Request
c2
Insurance Quote Service
information about new customer

(c) Active SR with direct request (new customer information)


Insurance Request Management Service
Process Insurance Request
d1
Request blacklist information
Insurance Quote Service
Blacklist Service
Blacklist information
d2

(d) Active SP with direct request and direct data transmission (blacklist information)


Insurance Request Management Service
Process Insurance Request
Insurance Quote Service
Sales Service
Base insurance quote
e1

(e) Active DP with direct data transmission (base insurance quote)

Figure 10: Coordination scenarios required to manage all data dependencies in the insurance case

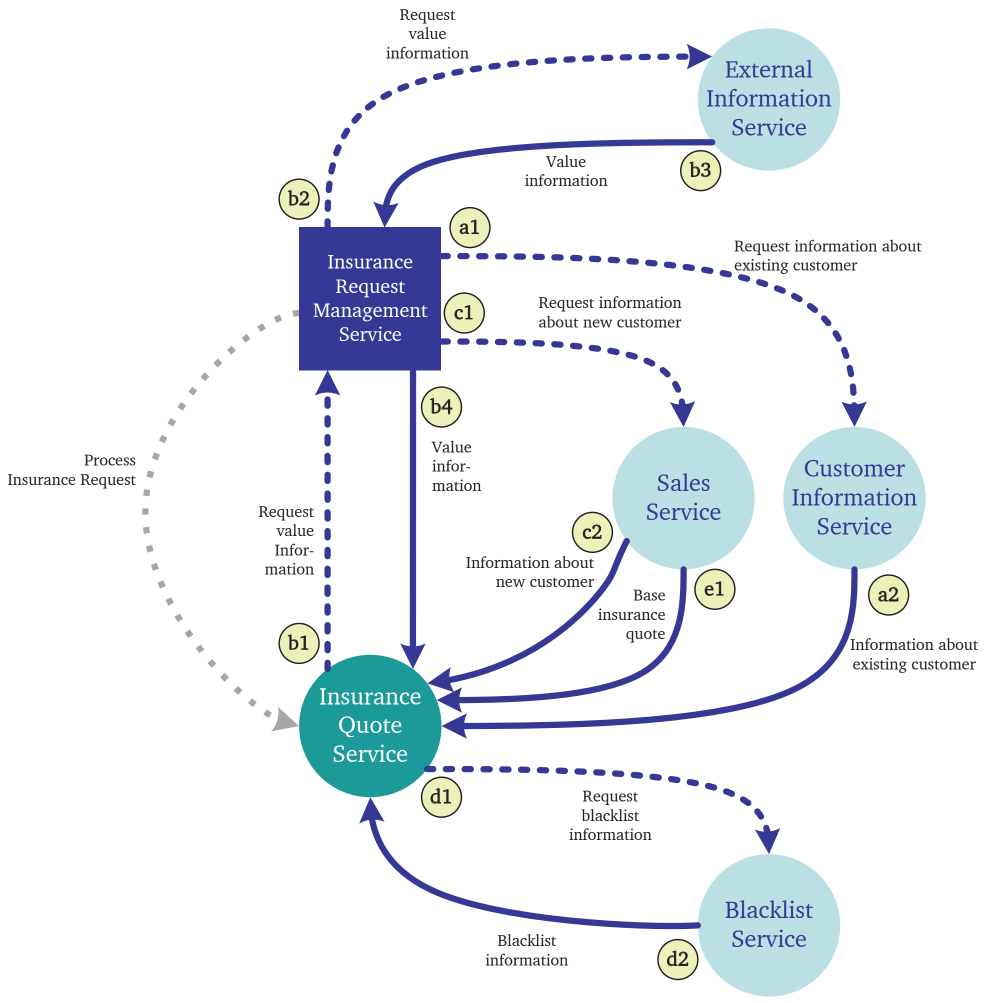

Request value information
External Information Service
Value information
b3
b2
a1
Insurance Request Management Service
Request information about existing customer
c1
Request information about new customer
b4
Value infor-mation
Process Insurance Request
Request value Infor-mation
Sales Service
Customer Information Service
c2
Information about new customer
e1
Base insurance quote
a2
Information about existing customer
b1
Insurance Quote Service
d1
Request blacklist information
Blacklist Service
Blacklist information
d2

Figure 11: Combining coordination scenarios from Figure 10(a) to 10(d) into a final solution

5.3. Implementation

In order to demonstrate the practical utility of the approach, the pattern language was implemented into a tool for pattern-based coordination in process-based service compositions. This tool also supports the pattern language for sequence dependencies, which helps service engineers to construct (partially) centralized or decentralized coordination scenarios for triggering of task execution in the appropriate order. Since the tool supports both pattern languages, it enables to design control logic and data flow separately, which means that, for example, a central control flow does not necessarily have to imply a central data flow. Before discussing the tool in detail, we briefly highlight the main ideas behind pattern-based service composition and coordination.

5.3.1. Pattern-based service composition and coordination

Pattern-based service composition and coordination starts from the idea that the services in a service composition support business process tasks. This means that a service needs to be able to receive business requests (e.g. for triggering the execution of a business process task) and send out event notifications about the execution of the business process task it supports (e.g. the completion of the task). Furthermore, a service supporting a business process also needs the ability to acquire all data that is necessary to execute this process. Moreover, if the business process task results into task data output, then the service is supposed to send out this data. All the control logic concerning a service supporting a business process task can be described in an orchestration (e.g. a BPEL process). A skeleton for this orchestration can be generated automatically based on the business process specification (e.g. a BPMN diagram). Hence, developers only need to add logic so that, for example, business requests are translated into the appropriate actions to start a business process execution.

The main advantage of this approach is that all orchestrations forming the global interaction and coordination scenario are automatically generated from a business process specification after selecting specific coordination patterns. Afterwards, it is relatively easy to apply other solutions for managing data (and sequence) dependencies so that a different coordination scenario is composed, e.g. because of a changed context with a different tradeoff of forces.

Since this article presents the pattern language for managing data dependencies, the discussion in this subsection is mainly about how the patterns allow one to automatically generate coordination scenarios that deal with data dependencies. However, designing coordination logic for process-based service compositions also requires the application of the pattern language for managing sequence dependencies. That pattern language is also supported by the tool. Additionally, the tool supports more complex coordination scenarios in which the required data is provided as the result of business task execution or in which input data is needed to make a decision on different possible branches for business process continuation. The presentation of these extra functionalities is however beyond the scope of this paper as this would require the description of the sequence coordination pattern language.

5.3.2. Tool implementation and its application in the insurance case study

Figure 12 visually summarizes the idea of pattern-based coordination by showing the input and output for the tool[5]. One should note that the tool is conceived as a proof-of-concept that demonstrates the feasibility of the approach rather than as a full-fledged BPEL generator. In its current implementation, little attention has been devoted to user friendliness, performance issues, etc. Nevertheless, the present architecture can be used as a basis for a more elaborated environment.

Below we will discuss each aspect by explaining how the tool can be used in the real-life insurance case described in Subsection 5.2.2.

The input for the tool consists of two parts. First, the tool requires input models that describe sequence and data dependencies. In a *BPMN model* the tool reads all sequence dependencies between business process tasks (e.g. a customers insurance request needs to be processed before an insurance contract is set up). Additionally, it can contain BPMN data input associations, which link BPMN data objects to BPMN tasks (OMG, 2010). Such a BPMN data input association implicitly defines a data dependency, because it specifies which data is required to execute a business process task (e.g. processing an insurance request requires blacklist information). As mentioned in Subsection 5.2.2 the data dependencies defined in the real-life insurance case can be situated in a larger business process that consists of tasks such as processing the customer's request, presenting an offer to the customer, making the contract,

[5]The source code of a prototype implementation can be downloaded from http://www.econ.kuleuven.be/geert.monsieur/public/phd/

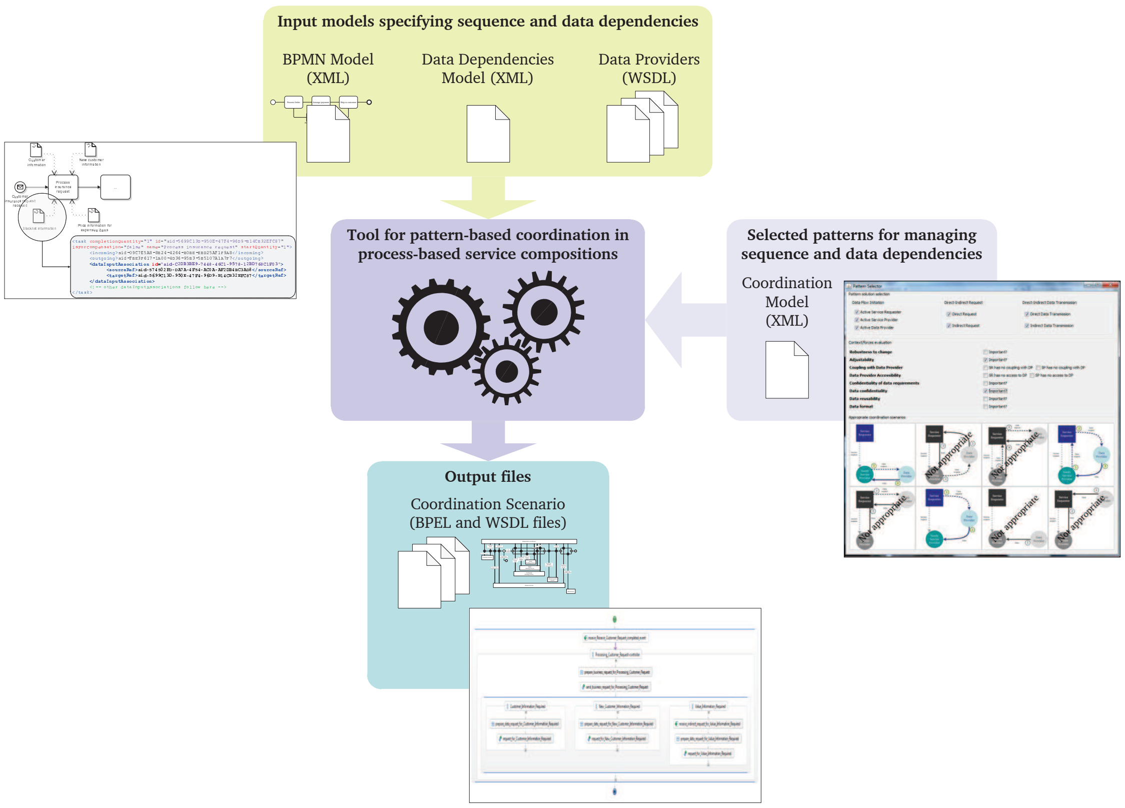
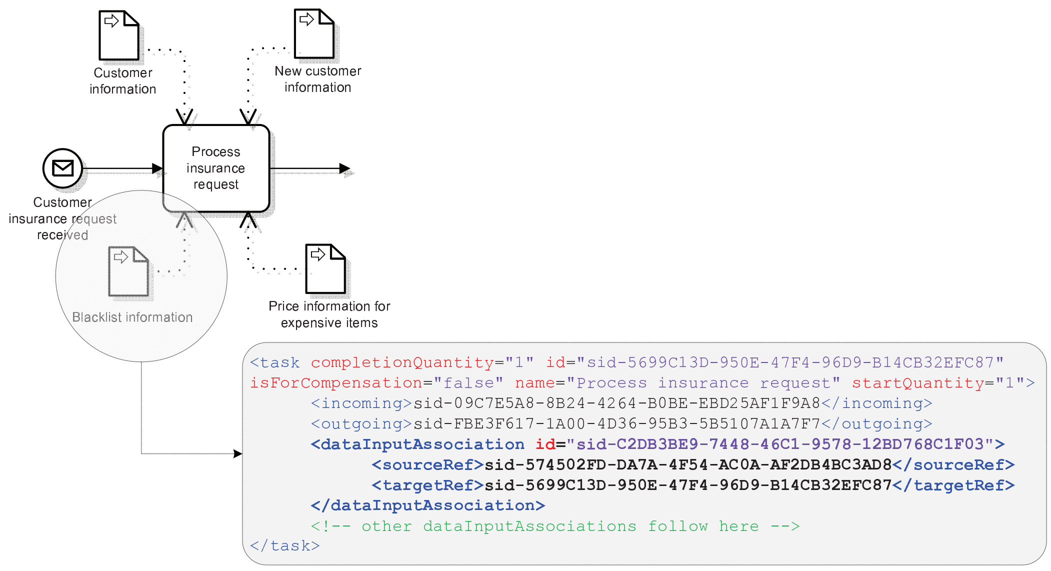

Input models specifying sequence and data dependencies
BPMN Model (XML)
Data Dependencies Model (XML)
Data Providers (WSDL)
Tool for pattern-based coordination in process-based service compositions
Selected patterns for managing sequence and data dependencies
Coordination Model (XML)
Not appropriate
Output files
Coordination Scenario (BPEL and WSDL files)

Figure 12: Input and output for pattern-based coordination


Customer information
New customer information
Process insurance request
Customer insurance request received
Blacklist information
Price information for expensive items

```
<task completionQuantity="1" id="sid-5699C13D-950E-47F4-96D9-B14CB32EFC87"
isForCompensation="false" name="Process insurance request" startQuantity="1">
      <incoming>sid-09C7E5A8-8B24-4264-B0BE-EBD25AF1F9A8</incoming>
      <outgoing>sid-FBE3F617-1A00-4D36-95B3-5B5107A1A7F7</outgoing>
      <dataInputAssociation id="sid-C2DB3BE9-7448-46C1-9578-12BD768C1F03">
            <sourceRef>sid-574502FD-DA7A-4F54-AC0A-AF2DB4BC3AD8</sourceRef>
            <targetRef>sid-5699C13D-950E-47F4-96D9-B14CB32EFC87</targetRef>
      </dataInputAssociation>
      <!-- other dataInputAssociations follow here -->
</task>
```

Figure 13: Partial BPMN model for insurance case study (including data input association)

sending the insurance policy to the customer, and payment by the customer. In this article we are only interested in the execution of the first task that is about processing a customer's request and the data dependencies that come with this task. Therefore, we can limit the business process description to one task that requires several data objects (see BPMN model in Figure 13). In this subsection we omitted the data requirement about the base insurance quote. This data dependency is managed using an ACTIVE DATA PROVIDER, which implies that the Insurance Quote Service is also involved in another process that receives updates on the base insurance quote. This coordination process is independent from the coordination process managing the other dependencies. Coordination processes with ACTIVE Data Providers are considered trivial and therefore not generated by the tool. In a separate XML file, the *data dependencies model* (see Figure 14), data dependencies are explicitly defined by linking the BPMN data input associations to Data Providers (e.g. the blacklist information that is required for processing an insurance request needs to be retrieved from the Blacklist service or data provider with id *DP4*). These Data Providers are also defined in the data dependencies model by specifying WSDL details (e.g. in Figure 14 it is specified that blacklist information can be requested from a data provider with id *DP4* by invoking the operation with the name *requestData* as defined in the WSDL file *BlacklistService.wsdl*). The second part of the input for the tool is the *coordination model* (XML file), which specifies which patterns need to be used to generate an executable coordination scenario. This model is the result of following the design guidelines and applying the evaluation criteria that are included in the patterns (see Subsection 5.2 on guided design). For each data dependency defined in the data dependencies model the coordination model specifies which pattern solutions are to be used in the generated code to manage the data dependency (e.g. the data dependency about blacklist information is managed using a ACTIVE SERVICE PROVIDER with DIRECT REQUEST and DIRECT DATA TRANSMISSION)[6].

Generated BPEL processes for the Service Requester and the Insurance Quote Service (i.e. Service Provider) are shown in Figure 16 and Figure 17. In order to test the correctness of the generated BPEL processes we deployed several test examples, including the example described in this section, to several instances of the OW2 Orchestra BPEL engine[7]. All generated BPEL processes were successfully executed.

6. Related work

Zirpins et al. (2004) propose to make a distinction between the *logical dependencies* that are modeled by the

[6]With respect to the sequence dependencies the coordination model specifies which BPMN tasks (as defined in the BPMN model) are to be coordinated by a single *coordinator* (i.e. a single BPEL process orchestrating multiple BPMN tasks). Since this article mainly focuses on data dependencies the coordination model shown in 15 only specifies one coordinator, executing one task (i.e. process insurances request).

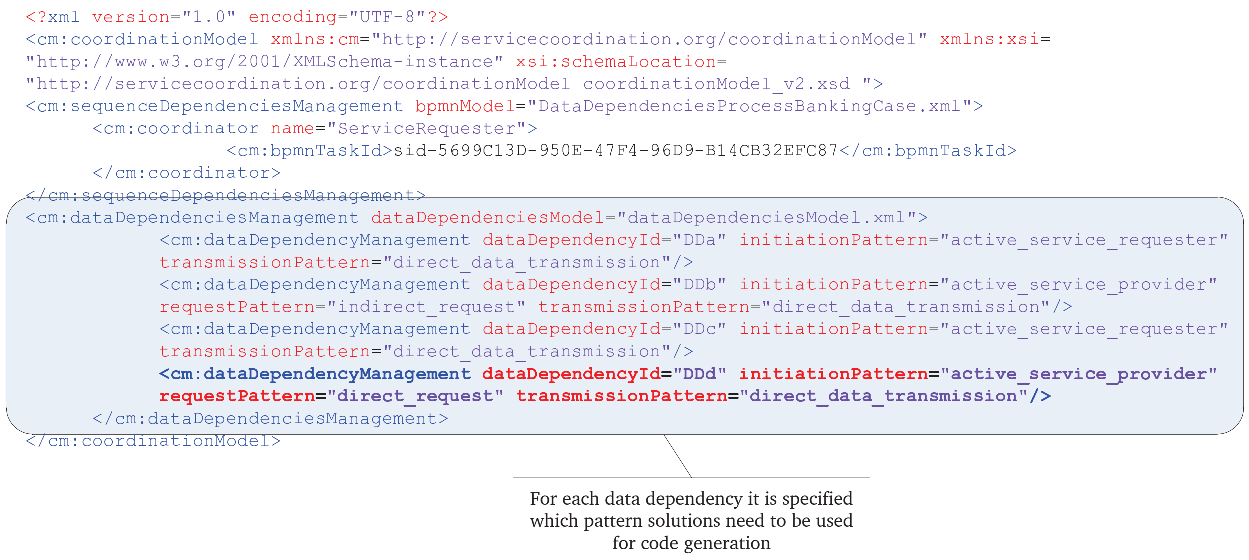
[7]http://orchestra.ow2.org/

Data Providers definitions (i.e. WSDL information on Data Providers)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<dd:dataDependenciesModel bpmnLocation="DataDependenciesProcessBankingCase.xml" xmlns:dd=
"http://servicecoordination.org/dataDependencies" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://servicecoordination.org/dataDependencies DataDependencies_v3.xsd ">
     <dd:dataProviders>
           <dd:dataProvider id="DP1" name="CRMService"
           partnerLinkTypeName="DataProviderLink"
           receiveOperationName="receiveData" receiveRoleName="DataReceiver"
           requestOperationName="requestData" requestRoleName="DataProvider"
           wsdlLocation="CRMService.wsdl"/>
           <dd:dataProvider id="DP2" name="ExternalInformationService"
           partnerLinkTypeName="DataProviderLink" receiveOperationName="receiveData"
           receiveRoleName="DataReceiver"
           requestOperationName="requestData" requestRoleName="DataProvider"
           wsdlLocation="ExternalInformationService.wsdl"/>
           <dd:dataProvider id="DP3" name="SalesService"
           partnerLinkTypeName="DataProviderLink"
           receiveOperationName="receiveData" receiveRoleName="DataReceiver"
           requestOperationName="requestData" requestRoleName="DataProvider"
           wsdlLocation="SalesService.wsdl"/>
           <dd:dataProvider id="DP4" name="BlacklistService"
           partnerLinkTypeName="DataProviderLink"
           receiveOperationName="receiveData" receiveRoleName="DataReceiver"
           requestOperationName="requestData" requestRoleName="DataProvider"
           wsdlLocation="BlacklistService.wsdl"/>
     </dd:dataProviders>
     <dd:dataDependencies>
           <dd:externalDataDependency bpmnTaskId="sid-5699C13D-950E-47F4-96D9-B14CB32EFC87"
           dataInputAssociationId="sid-C2DB3BE9-7448-46C1-9578-12BD768C1F03"
           dataProviderId="DP1" id="DDa" name="Customer_Information_Required"/>
           <dd:externalDataDependency bpmnTaskId="sid-5699C13D-950E-47F4-96D9-B14CB32EFC87"
           dataInputAssociationId="sid-C2DB3BE9-7448-46C1-9578-12BD768C1F03"
           dataProviderId="DP2" id="DDb" name="Value_Information_Required"/>
           <dd:externalDataDependency bpmnTaskId="sid-5699C13D-950E-47F4-96D9-B14CB32EFC87"
           dataInputAssociationId="sid-C2DB3BE9-7448-46C1-9578-12BD768C1F03"
           dataProviderId="DP3" id="DDc" name="New_Customer_Information_Required"/>
           <dd:externalDataDependency bpmnTaskId="sid-5699C13D-950E-47F4-96D9-B14CB32EFC87"
           dataInputAssociationId="sid-C2DB3BE9-7448-46C1-9578-12BD768C1F03"
           dataProviderId="DP4" id="DDd" name="Blacklist_Details_Required"/>
     </dd:dataDependencies>
</dd:dataDependenciesModel>
```

Excplicit Data Dependencies definitions (i.e. links between BPMN dataInputAssociations and Data Providers defined above)

Figure 14: Data Dependencies model specifying Data Providers and data dependencies



```
<?xml version="1.0" encoding="UTF-8"?>
<cm:coordinationModel xmlns:cm="http://servicecoordination.org/coordinationModel" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://servicecoordination.org/coordinationModel coordinationModel_v2.xsd ">
<cm:sequenceDependenciesManagement bpmnModel="DataDependenciesProcessBankingCase.xml">
      <cm:coordinator name="ServiceRequester">
                  <cm:bpmnTaskId>sid-5699C13D-950E-47F4-96D9-B14CB32EFC87</cm:bpmnTaskId>
      </cm:coordinator>
</cm:sequenceDependenciesManagement>
<cm:dataDependenciesManagement dataDependenciesModel="dataDependenciesModel.xml">
            <cm:dataDependencyManagement dataDependencyId="DDa" initiationPattern="active_service_requester"
            transmissionPattern="direct_data_transmission"/>
            <cm:dataDependencyManagement dataDependencyId="DDb" initiationPattern="active_service_provider"
            requestPattern="indirect_request" transmissionPattern="direct_data_transmission"/>
            <cm:dataDependencyManagement dataDependencyId="DDc" initiationPattern="active_service_requester"
            transmissionPattern="direct_data_transmission"/>
            <cm:dataDependencyManagement dataDependencyId="DDd" initiationPattern="active_service_provider"
            requestPattern="direct_request" transmissionPattern="direct_data_transmission"/>
      </cm:dataDependenciesManagement>
</cm:coordinationModel>
```

For each data dependency it is specified which pattern solutions need to be used for code generation

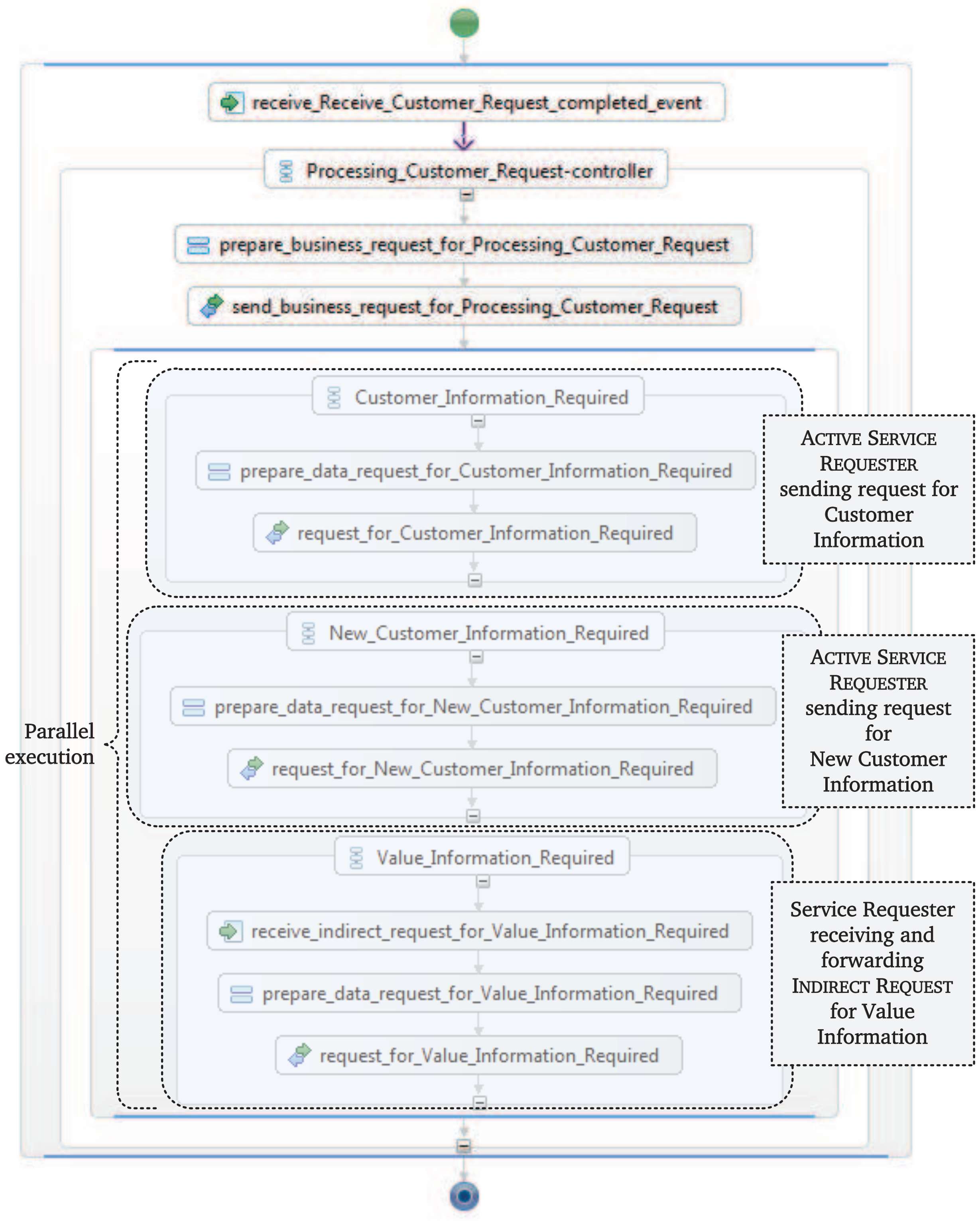
Figure 15: Coordination model specifying selected coordination patterns

interaction logic and the *operational coordination* that refers to the procedure or method that is utilized to enforce the logical dependencies. We make a distinction between sequence and data dependencies. Similarly, the operational coordination matches our vision on coordination, which is about managing the sequence and data dependencies. Zirpins et al. (2004) argued that while workflow processes represent the logical dependencies of interactions (i.e. causal and data relationships of message exchanges) they often simultaneously act as instructions for their coordination on the execution-level by distributed workflow management systems. As such, the coordination procedure emerges only implicitly as a side-effect of dependencies from the interaction logic and not because of application-specific reasons.

However, there are in most cases multiple alternatives for the enforcement of the abstract interaction logic. Therefore Zirpins et al. (2004) suggest that a technical solution for service composition should consist of a combination of design and implementation patterns. A *design pattern* corresponds to the interaction logic that only specifies the generic process characteristics, while an *implementation pattern* refers to the refinement of the interaction logic that is needed for the concrete coordination of services. Zirpins et al. (2004) state that the criteria for the choice of the most appropriate coordination pattern must be specified by so called coordination policies. A *coordination policy* describes the effect of a coordination variant in terms of specific (non-functional) service properties and thereby controls the choice of alternatives. In summary, we can conclude that in this article we are looking for both implementation patterns and coordination policies (Zirpins et al., 2004; Zirpins and Lamersdorf, 2004). The implementation patterns allow us to systematically construct coordination scenarios, and the coordination policies make it possible to construct the most appropriate coordination scenario in a certain business context.

Barros et al. (2005) have proposed a set of *service interaction patterns*, aiming to consolidate recurrent interaction scenarios in orchestrations and choreographies, and abstract them in a way that provides reusable knowledge. Furthermore, the service interaction patterns are intended for assessing an orchestration or choreography language for its interaction modeling capabilities. In the past such evaluations were conducted for BPEL (Barros et al., 2005), WS-CDL (Decker et al., 2006), BPMN (Decker and Puhlmann, 2007; Decker and Barros, 2008) and BPEL4Chor (Decker et al., 2007). Although the service interaction patterns may be composed through operators expressing flow dependencies (e.g. sequence, choice, etc.) (Barros and Börger, 2005), no guidelines exist on how to combine the patterns to construct coordination logic that manage data dependencies.

Zdun et al. (2006) have proposed a pattern-based architectural framework for SOAs. In their reference architecture the coordination logic in service compositions is referred to as the *process integration logic*. Since both a business process and coordination logic are represented using process flows, Zdun et al. (2006) propose to make a distinction between two general types of process flows: *macroflow* representing the higher-level business process, and microflow



receive_Receive_Customer_Request_completed_event
Processing_Customer_Request-controller
prepare_business_request_for_Processing_Customer_Request
send_business_request_for_Processing_Customer_Request
Customer_Information_Required
prepare_data_request_for_Customer_Information_Required
request_for_Customer_Information_Required
ACTIVE SERVICE REQUESTER sending request for Customer Information
New_Customer_Information_Required
prepare_data_request_for_New_Customer_Information_Required
request_for_New_Customer_Information_Required
ACTIVE SERVICE REQUESTER sending request for New Customer Information
Parallel execution
Value_Information_Required
receive_indirect_request_for_Value_Information_Required
prepare_data_request_for_Value_Information_Required
request_for_Value_Information_Required
Service Requester receiving and forwarding INDIRECT REQUEST for Value Information

Figure 16: Generated BPEL process for the Service Requester

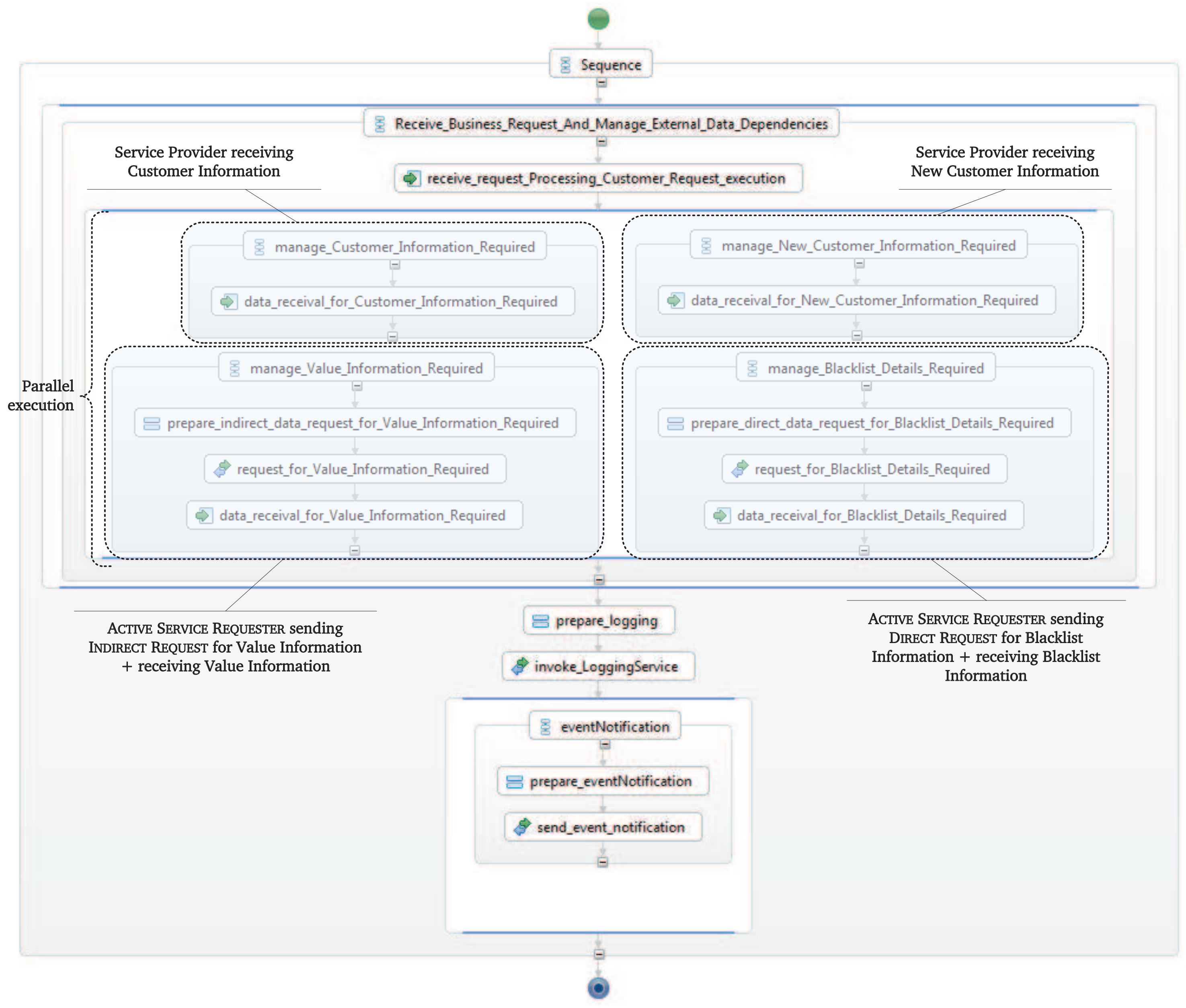
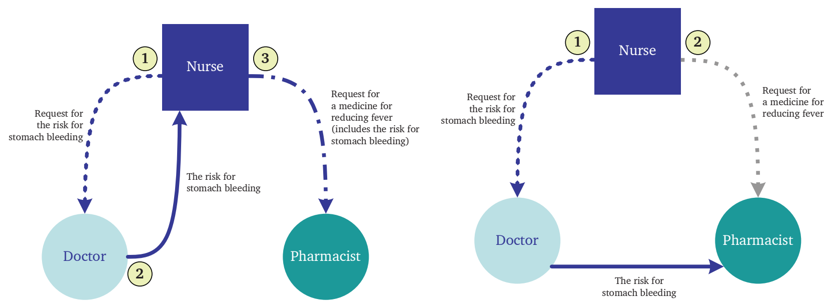

Sequence
Receive_Business_Request_And_Manage_External_Data_Dependencies
Service Provider receiving Customer Information
receive_request_Processing_Customer_Request_execution
Service Provider receiving New Customer Information
manage_Customer_Information_Required
data_receival_for_Customer_Information_Required
manage_New_Customer_Information_Required
data_receival_for_New_Customer_Information_Required
Parallel execution
manage_Value_Information_Required
prepare_indirect_data_request_for_Value_Information_Required
request_for_Value_Information_Required
data_receival_for_Value_Information_Required
manage_Blacklist_Details_Required
prepare_direct_data_request_for_Blacklist_Details_Required
request_for_Blacklist_Details_Required
data_receival_for_Blacklist_Details_Required
Active Service Requester sending Indirect Request for Value Information + receiving Value Information
prepare_logging
invoke_LoggingService
Active Service Requester sending Direct Request for Blacklist Information + receiving Blacklist Information
eventNotification
prepare_eventNotification
send_event_notification

Figure 17: Generated BPEL process for the Service Provider (i.e. insurance quote service that processes customer requests



1
Nurse
3
Request for the risk for stomach bleeding
Request for a medicine for reducing fever (includes the risk for stomach bleeding)
The risk for stomach bleeding
Doctor
2
Pharmacist
1
Nurse
2
Request for the risk for stomach bleeding
Request for a medicine for reducing fever
Doctor
Pharmacist
The risk for stomach bleeding

(a) Central data flow (b) Decentral data flow

Figure 18: Two possible data flows for hospital example

addressing the process flow within a macroflow activity. The distinction between micro- and macroflow is a conceptual decision in order to be able to design process steps at the right level of granularity when designing at the long running business process level (macroflow) or the short running, more technical level (microflow) (Hentrich and Zdun, 2006). Typically, a microflow consists of coordinated service interactions. However, no patterns were referenced for constructing such service interactions systematically, nor were patterns referenced for different sorts of microflow (i.e. templates or coordination styles).

Data dependencies are related to the *data flow* concept in service compositions. In general, data flow can be defined as the service interactions that are necessary for sending data from one service that can provide certain data to another service that needs that data (Barker et al., 2008b; Yang, 2003; Weber et al., 2003; Charfi and Mezini, 2007). A data flow thus specifies how data dependencies are managed. Therefore, we also compare our work to studies that specifically contain approaches to identify data dependencies and realise data flows (possibly proposing different alternatives) in a service composition. We illustrate the results of these studies by means of the running example presented in Section 2. We will show that these studies do not cover all aspects that are important for the design of an appropriately coordinated service-based system.

In the descriptions below, we use two coordination scenarios for the hospital example, which are represented in figures 18(a) and 18(b). Each arrow corresponds to a message sent between two entities. The dashed arrows refer to service invocations, while the solid arrows denote the transfer of data between two entities. The semi-dashed arrow (as used in figure 18(a)) indicates that the data is included in the invocation message. While in figure 18(a) all data passes via the nurse (central data flow), the data flow in figure 18(b) is decentral, since data flows directly from one service to the other. As we will show below, the contributions of many studies can be easily explained by means of this small example consisting of two possible coordination scenarios.

Barker et al. (2008b) and Barker et al. (2008a) have presented a Web services based architecture that allows centralizing component invocations (centralized control flow) and decentralizing data flows (similar to figure 18(b)). This architecture consists of a centralized orchestration engine that issues control flow messages to Web services taking part in service composition. However, enrolled Web services can pass data messages among themselves, as in a peer to peer model. The architecture is mainly based on the idea of so called *proxies*, which are deployed in the vicinity of Web services. These proxies realize the more efficient data flow between component services.

Liu et al. (2002a,b) have published a mathematical model that is built to compare the data flow performances. They concluded that decentralized data flow is in general superior in performance (i.e. the service composition in figure 18(b) outperforms 18(a)). Subsequently, they developed a Flow-based Infrastructure for Composing Autonomous Services (FICAS) (Liu et al., 2002a,b). Autonomous services are built to support the service access protocol, which enforces the explicit separation of data flows from control flows. In FICAS the so called autonomous services are implemented by wrapping each software application or service into an autonomous service with a mediator.

The infrastructure based on so called service invocation triggers, introduced by Binder et al. (2006), is very similar to FICAS. In this infrastructure service invocation triggers also act as proxies for individual service invocations. Triggers collect the required input data before they invoke the service. Moreover, they forward service outputs to exactly those services that need the output. In order to make use of triggers, business processes are decomposed into sequential fragments, and the data dependencies are encoded within the triggers. Once the trigger of the first service in a business process has received all input data, the execution of that service is started and the outputs are forwarded to the triggers of subsequent services. Consequently, the service composition is implemented in a fully decentralized way, the data is transmitted directly from the producer to all consumers.

Balasooriya et al. (2005) use the same ideas for decentralizing data flows. In particular, they create a proxy wrapper around each Web service. The proxy wrappers embed the coordination logic so that instances of wrapped Web services become stateful self-coordinating web objects. However, the proxy wrappers need to interact with the actual Web service to complete each method invocation.

We can conclude that several approaches exists that cater for alternative data flows. The proposed architectural infrastructures for such data flows often use the same idea: wrapping each component service with additional logic that decides where to send input or output data. Obviously, these infrastructures are valuable and useful when one wants to implement a specific data flow. However, the focus on the problem of designing the data flow itself is rather limited. Furthermore, as we will show below, it remains difficult for a service composer to construct a well coordinated service composition. There are two main reasons why the current approaches are not entirely adequate for this purpose:

1 As most approaches have only a limited focus on designing the data flow, these studies fail to systematically analyze the coordination problem. Most approaches allow finding alternative data flows, but do not provide a systematic way of building different coordination scenarios nor do they analyze the advantages and disadvantages of alternatives. As a consequence, they fail to exhaustively identify all possible coordination scenarios. The approaches mostly propose techniques for decentralizing data flows in service compositions. Applied to the hospital example, this would mean that a scenario such as shown in figure 18(a) can be transformed into a scenario such as shown in figure 18(b). However, one can easily see that there are more possibilities. For example, the scenario represented in figure 19 contains a different coordination scenario. In this scenario the pharmacist requests and receives the risk information directly from the doctor, which can be considered as yet another different way of managing the data dependency between the pharmacist and the doctor. Other possible scenarios were illustrated in the section discussing the running example (see Figures 2(a) and 2(b)).

2 The main motivation behind existing approaches are performance issues (i.e. communication overhead, etc.). Only the work by Balasooriya et al. (2005) recognizes that decentral data flow can be required due to security, privacy, or licensing imperatives. However, when evaluating their infrastructure, they only focus on the performance aspect. To the best of our knowledge, no studies about data dependency management take into account other aspects that could influence the choice of a specific data flow solution such as data confidentiality, loose coupling or robustness to change. This can result in badly or suboptimally coordinated service compositions and service-based systems. For example, the pharmacist and doctor in the decentralized data flow scenario shown in figure 2(b) are not optimally coordinated. This is due to the fact that nurses probably should not need to understand which data is required by the pharmacist. Nurses simply want to consume the pharmacist's services, and it is to be avoided that changes in data requirements on behalf of the pharmacist result in changes in how nurses need to work (or consume the pharmacist's services). Hence, the scenarios represented in figures 2(a) and 19 are probably more appropriate, because in these scenarios the nurse does not have to know which data is needed by the pharmacist. This example illustrates that robustness to change is another useful criterion to be considered next to performance issues.

7. Conclusion

When services are composed into service-based systems, the design of a data flow is a crucial part of the service coordination. This often occurs in an ad hoc fashion without any tool support (Papazoglou, 2005; Papazoglou and Van den Heuvel, 2007). The pattern language and tool presented in this article provide a systematic way of composing coordination scenarios from fundamental building blocks so that the coordination scenario manages all data dependencies.

While related work on the (alternative) design of data flows in service compositions is mainly focused on optimizing the performance, the pattern-based approach proposed in this paper analyzes the advantages and disadvantages of each design alternative by considering *multiple* evaluation criteria or pattern forces, including robustness to change, loose coupling and data confidentiality. The approach divides the problem of managing a data dependency into three

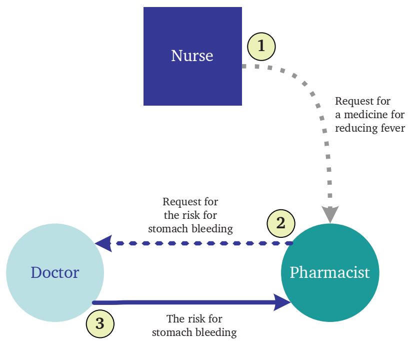

Nurse
1
Request for a medicine for reducing fever
Request for the risk for stomach bleeding
2
Doctor
Pharmacist
3
The risk for stomach bleeding

Figure 19: An alternative data flow for the scenarios represented in figures 18(a) and 18(b)

subproblems, each addressed by a separate pattern. In each solution presented in a particular pattern forces are balanced differently. By giving a weight to the forces, service composers can be guided towards the most appropriate coordination scenario in the context at hand.

In Section 3 three objectives were defined that were used to evaluate the pattern language in Section 5. First, we demonstrated the practical utility of the pattern language as guidelines for developers by applying it in a real-life insurance case. As the case involves a large set of data to which different forces apply in different ways and multiple data providers, it demonstrated the practical utility in complex real life situations that require multiple applications of the pattern language. Second, we showed that *all* potential coordination scenarios for managing a data dependency can be composed by combining the patterns of the pattern language. The fact that the pattern language presented in this paper is complete, has important consequences for the way in which coordination logic is constructed in the future. In the past researchers argued that an over-emphasis on service interactions is at the expense of other aspects like business goals (Ko et al., 2009; Koubarakis and Plexousakis, 1999; Andersson et al., 2005). However, since all potential coordination scenarios can be composed from the patterns, it makes no sense anymore to spend expensive time designing coordination scenarios at the level of service interactions and message exchanges. Third, the patterns proved to form a good basis for configurable model-to-code transformations (Zimmermann et al., 2006), making it possible to automatically generate executable BPEL coordination scenarios. In line with the principles of the Model-Driven Architecture (Kleppe et al., 2003) it contributes to an efficient (e.g. automatically generated coordination scenarios) and effective (e.g. less errors and more consistency) development of service composition. Developers do not need to repeatedly re-encode the same implementation patterns, resulting in an increased reuse. The patterns presented in this paper were successfully implemented in a tool for pattern-based service composition and coordination. This showed that the patterns can be combined with the patterns for managing sequence dependencies, which makes it possible to design data flows independently from the control flow (e.g. central control flow combined with a decentral data flow).

The core results presented in this paper were written as patterns constituting a pattern language. The use of patterns as a description technique immensely facilitates the communication of the research results. Although we have successfully applied the patterns to automatically generate coordination logic from business process specifications, people are the prime audience for patterns. Patterns form a specialized but common vocabulary that software architects and developers can use to discuss particular problems that arise in their projects, resulting into a better joint understanding of specific problems and solutions to these problems (Buschmann et al., 2007). Furthermore, previous research has shown that patterns improve the repeatability, usability and reuse of design practices (Ng et al., 2006; Prechelt et al., 2001).

References

Andersson, B., Bider, I., Johannesson, P., Perjons, E., 2005. Towards a formal definition of goal-oriented business process patterns. Business Process Management Journal 11, 650–662.

Balasooriya, J., Padhye, M., Prasad, S.K., Navathe, S.B., 2005. Bondflow: A system for distributed coordination of workflows over web services, in: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2005) - Workshop 1, IEEE Computer Society, Washington, DC, USA. p. 121.1.

Barker, A., Weissman, J.B., Van Hemert, J., 2008a. Eliminating the middleman: peer-to-peer dataflow, in: Proceedings of the 17th international symposium on High performance distributed computing (HPDC 2008), ACM, New York, NY, USA. pp. 55–64.

Barker, A., Weissman, J.B., Van Hemert, J., 2008b. Orchestrating data-centric workflows, in: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2008), IEEE Computer Society, Washington, DC, USA. pp. 210–217.

Barros, A., Börger, E., 2005. A compositional framework for service interaction patterns and interaction flows, in: Lau, K.K., Banach, R. (Eds.), Formal Methods and Software Engineering. Springer-Verlag Berlin Heidelberg. volume 3785 of *Lecture Notes in Computer Science*, pp. 5–35.

Barros, A., Dumas, M., ter Hofstede, A.H., 2005. Service Interaction Patterns, in: Van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (Eds.), Business Process Management. Springer-Verlag Berlin Heidelberg. volume 3649 of *Lecture Notes in Computer Science*, pp. 302–318.

Binder, W., Constantinescu, I., Faltings, B., 2006. Decentralized orchestration of composite web services, in: Proceedings of the IEEE International Conference on Web Services (ICWS 2006), IEEE Computer Society, Washington, DC, USA. pp. 869–876.

Buschmann, F., Henney, K., Schmidt, D., 2007. Pattern-oriented software architecture: On patterns and pattern languages. John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, England.

Charfi, A., Mezini, M., 2007. AO4BPEL: An Aspect-oriented Extension to BPEL. World Wide Web 10, 309–344.

Clocksin, W., Mellish, C., 2003. Programming in PROLOG (Fifth Edition). Springer-Verlag Berlin Heidelberg, New York, NY, USA.

Decker, G., Barros, A., 2008. Interaction modeling using bpmn, in: Proceedings of the 2007 international conference on Business Process Management (BPM 2007), Springer-Verlag Berlin Heidelberg. pp. 208–219.

Decker, G., Kopp, O., Leymann, F., Weske, M., 2007. BPEL4Chor: Extending BPEL for Modeling Choreographies, in: Proceedings of IEEE 2007 International Conference on Web Services (ICWS 2007), IEEE Computer Society, Washington, DC, USA. pp. 296 –303.

Decker, G., Overdick, H., Zaha, J., 2006. On the Suitability of WS-CDL for Choreography Modeling, in: Proceedings of Methoden, Konzepte und Technologien für die Entwicklung von dienstebasierten Informationssystemen (EMISA 2006), Citeseer.

Decker, G., Puhlmann, F., 2007. Extending bpmn for modeling complex choreographies, in: Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems (OTM 2007), Springer-Verlag Berlin Heidelberg. pp. 24–40.

Erl, T., 2007. SOA Principles of Service Design. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design patterns: elements of reusable object-oriented software. Addison-wesley Reading, MA.

Goethals, F., 2008. Important Issues for Evaluating Inter-Organizational Data Integration Configurations. Electronic Journal Information Systems Evaluation 11, 185–196.

Habala, O., Simo, B., Gatial, E., Hluchy, L., 2008. Automatic data reuse in grid workflow composition, in: Proceedings of the 8th international conference on Computational Science, Part I, Springer-Verlag, Berlin, Heidelberg. pp. 194–202.

Haesen, R., De Rore, L., Snoeck, M., Lemahieu, W., Poelmans, S., 2006. Active-passive hybrid data collection, in: Proceedings of the 11th European Conference on Pattern Languages of Programs (EuroPLoP 2006), pp. 565–577.

Hentrich, C., Zdun, U., 2006. Patterns for process-oriented integration in service-oriented architectures, in: Proceedings of the 11th European Conference on Pattern Languages of Programs (EuroPLoP 2006).

Hevner, A.R., March, S.T., Park, J., Ram, S., 2004. Design science in information systems research. MIS Quarterly 28, 75–105.

Janssen, M., Feenstra, R., 2008. Socio-technical design of service compositions: a coordination view, in: Proceedings of the 2nd International Conference on Theory and Practice of Electronic Governance (ICEGOV 2008), ACM, New York, NY, USA. pp. 323–330.

Kleppe, A.G., Warmer, J., Bast, W., 2003. MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Ko, R.K.L., Lee, S.S.G., Lee, E.W., 2009. Business process management (BPM) standards: a survey. Business Process Management Journal 15, 744–791.

Koubarakis, M., Plexousakis, D., 1999. Business process modelling and design - a formal model and methodology. BT Technology Journal 17, 23–35.

Legner, C., Vogel, T., 2007. Design principles for b2b services - an evaluation of two alternative service designs, in: Proceedings of IEEE International Conference on Services Computing 2007 (SCC 2007), pp. 372–379.

Liu, D., Law, K.H., Wiederhold, G., 2002a. Analysis of integration models for service composition, in: Proceedings of the 3rd international workshop on Software and performance (WOSP 2002), ACM, New York, NY, USA. pp. 158–165.

Liu, D., Law, K.H., Wiederhold, G., 2002b. Data-flow distribution in FICAS service composition infrastructure, in: Proceedings of the 15th International Conference on Parallel and Distributed Computing Systems (PDCS 2002), ISCA, Louisville, Kentucky USA.

Malone, T., Crowston, K., 1994. The interdisciplinary study of coordination. ACM Computing Surveys (CSUR) 26, 119.

Metzger, A., Pohl, K., 2009. Towards the Next Generation of Service-Based Systems: The S-Cube Research Framework, in: van Eck, P., Gordijn, J., Wieringa, R. (Eds.), Advanced Information Systems Engineering. Springer-Verlag Berlin Heidelberg. volume 5565 of *Lecture Notes in Computer Science*, pp. 11–16.

Monsieur, G., 2010. Pattern-based Coordination in Process-based Service Compositions. Ph.D. thesis. Faculty of Business and Economics, Katholieke Universiteit Leuven.

Monsieur, G., Snoeck, M., Lemahieu, W., 2010. Managing sequence dependencies in service compositions, in: Proceedings of the 15th European Conference on Pattern Languages of Programs (EuroPLoP 2010).

Ng, T.H., Cheung, S.C., Chan, W.K., Yu, Y.T., 2006. Work experience versus refactoring to design patterns: a controlled experiment, in: Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT '06/FSE-14), ACM, New York, NY, USA. pp. 12–22.

OASIS, 2007. Web Services Business Process Execution Language (WS-BPEL) Version 2.0. OASIS Standard.

OMG, 2010. Business Process Model and Notation (BPMN) Version 2.0. OMG Document (dtc/2010-06-05).

Paci, F., Ouzzani, M., Mecella, M., 2008. Verification of access control requirements in web services choreography, in: Proceedings of the 2008 IEEE International Conference on Services Computing - Volume 1, IEEE Computer Society, Washington, DC, USA. pp. 5–12.

Paikens, A., Arnicans, G., 2008. Use of Design Patterns in PHP-Based Web Application Frameworks. Scientific Papers University of Latvia, Computer Science and Information Technologies 733, 53–71.

Papazoglou, M., 2005. Extending the service-oriented architecture. Business Integration Journal 7, 18–21.

Papazoglou, M., Delis, A., Bouguettaya, A., Haghjoo, M., 1997. Class library support for workflow environments and applications. IEEE Transactions on Computers 46, 673–686.

Papazoglou, M., Van den Heuvel, W.J., 2007. Service oriented architectures: approaches, technologies and research issues. The VLDB Journal - The International Journal on Very Large Data Bases 16, 415.

Peffers, K., Tuunanen, T., Rothenberger, M., Chatterjee, S., 2007. A design science research methodology for information systems research. Journal of Management Information Systems 24, 45–77.

Prechelt, L., Unger, B., Tichy, W., Brössler, P., Votta, L., 2001. A controlled experiment in maintenance comparing design patterns to simpler solutions. IEEE Transactions on Software Engineering , 1134–1144.

Rising, L., 1998. The patterns handbook: Techniques, strategies, and applications, Cambridge University Press. volume 13 of *SIGS reference library series*.

Rising, L., 1999. Patterns: a way to reuse expertise. IEEE communications magazine 37, 34–36.

RosettaNet, n.d. RosettaNet Partner Interface Processes©(PIPs©). RosettaNet Standard.

W3C, 2005. Web Services Choreography Description Language (WS-CDL) Version 1.0. W3C Candidate Recommendation.

Weber, R., Schuler, C., Neukomm, P., Schuldt, H., Schek, H.J., 2003. Web service composition with O'GRAPE and OSIRIS, in: Proceedings of the 29th international conference on Very large data bases (VLDB 2003), VLDB Endowment. pp. 1081–1084.

Wielemaker, J., 2003. An overview of the SWI-Prolog programming environment, in: Proceedings of the 13th International Workshop on Logic Programming Environments, Department of Computer Science, K.U.Leuven, Leuven, Belgium. pp. 1–16.

Yang, J., 2003. Web service componentization. Communications of the ACM 46, 35–40.

Yang, J., Papazoglou, M., Van den Heuvel, W.J., 2002. Tackling the Challenges of Service Composition in E-Marketplaces, in: Proceedings of the 12th International Workshop on Research Issues in Data Engineering: Engineering E-Commerce/E-Business Systems (RIDE 2002), IEEE Computer Society, Washington, DC, USA. p. 125.

Zdun, U., Hentrich, C., Van der Aalst, W.M., 2006. A survey of patterns for service-oriented architectures. International Journal of Internet Protocol Technology 1, 132–143.

Zimmermann, O., Koehler, J., Leymann, F., 2006. The role of architectural decisions in model-driven soa construction, in: Proceedings of the 4th International Workshop on SOA and Web Services, colocated 21st International Conference on Object-Oriented Programming, Systems, Languages, and Applications.

Zirpins, C., Lamersdorf, W., 2004. Service Co-operation Patterns and their Customised Coordination, in: Proceedings of the Second European Workshop on Object Orientation and Web Service (EOOWS 2004).

Zirpins, C., Lamersdorf, W., Baier, T., 2004. Flexible coordination of service interaction patterns, in: Proceedings of the 2nd international conference on Service oriented computing (ICSOC 2004), ACM, New York, NY, USA. pp. 49–56.

Appendix A. Pattern forces (evaluation criteria)

EC1 *Robustness to change*: In a service-oriented environment it is critical that the propagation of changes due to the modification of the interface of a Service Provider is minimized. Consumers prefer to rely on a Service Provider that only rarely changes its interface. A change in the data requirements should minimally change the way the Service Provider is consumed. This evaluation criterion is related to the service design principle called service reusability, because a service can be considered more reusable if it has a relatively simple and stable interface (Erl, 2007).

EC2 *Adjustability*: A specific coordination scenario consists of a set of service interactions so that data available at the Data Provider is sent to the Service Provider. This criterion is about the ability to change which data is sent to the Service Provider in function of a specific service request. For example, in coordination scenarios to manage the data dependency between the pharmacist and the doctor this criterion can be used to make a distinction between coordination scenarios in which information regarding a *specific* patient is sent to the pharmacist and coordination scenarios in which information regarding *multiple* patients are sent to the pharmacist. Depending on the specific business context, a certain level of adjustability can be desired. For example, in the hospital setting efficiency issues can motivate coordination scenarios with high adjustability, so that pharmacists only receive information they really need (e.g. patient specific information instead of information regarding multiple patients). This evaluation criterion is derived from the service design principle called service granularity (Legner and Vogel, 2007; Erl, 2007). Low adjustability implies a fine-grained data transmission, while high adjustability results into more coarse-grained data exchanges.

EC3 *Coupling with Data Provider*: In some situations the data needed is not always provided by the same Data Provider. Each time a service takes over the role of Data Provider the party that is sending data requests to the Data Provider needs to be notified and modified properly. Similarly, each change in the interface of the Data Provider, requires a change in the implementation of the party that is interacting with the Data Provider.

Therefore, a common principle in service design called loose coupling is often applied (Erl, 2007). This means that preferably a Service Provider's implementation does not have to rely on several other services.

EC4 *Data provider accessibility*: Sometimes it is possible that the Service Provider does not know *which* service can provide the required data (e.g. the pharmacist does not know who is the patient's doctor). In other cases it can occur that the Service Provider does not have *access* to the specific Data Provider (e.g. the pharmacist does not have the phone number of the patient's doctor or is not allowed to call the doctor directly). However, it can also occur that only the Service Provider has access to the right Data Provider (e.g. only the pharmacist can request information concerning a potential risk for stomach bleeding). This evaluation criterion is inspired by research on the verification of access control requirements in choreographies (e.g. Paci et al. (2008)).

EC5 *Confidentiality of data requirements*: In order to complete its internal processing, a Service Provider needs data. It can occur that these data requirements are confidential (e.g. suppose that nurses cannot have insight into the pharmacist's internal decision processes), which means that only a limited set of services or even only the Service Provider itself knows which data is needed in the process of delivering its service. This evaluation criterion is related to the service design principle called service abstraction, because it is about hiding information about the Service Provider's data requirements (Erl, 2007).

EC6 *Data confidentiality*: When requesting a Data Provider to send the required data to an entity, it is important to realize that the provided data can be confidential and therefore there can exist a need to limit the number of entities that the Data Provider can share the data with (Goethals, 2008). For example, a Data Provider can demand that the provided data is only sent to the entity (e.g. a Service Provider) that needs the data and that it cannot be shared with other Service Providers or the Service Requester.

EC7 *Data reusability*: In some business cases data provided by a Data Provider is used by more than one Service Provider. In such situations an optimal coordination scenario limits the number of data requests that are sent to the Data Provider. This evaluation criterion is often used in grid workflow composition (Habala et al., 2008).

EC8 *Data format*: When the Data Provider replies, the data that is provided is possibly not in a form that is expected by the Service Provider. For example, the data format needs to be adapted, or the data should be made anonymous. In short, in some cases data transformations are desirable before the data is received by the Service Provider. Dealing with different data formats is a common challenge when information is shared among services (Goethals, 2008).

Appendix B. Pattern consequences

Appendix B.1. Data flow initiation

EC1 *Robustness to change*: In case of an ACTIVE SERVICE REQUESTER or an ACTIVE DATA PROVIDER every change in the Service Provider's data requirements results in a change in the implementation of the Service Requester or Data Provider. In contrast, in the ACTIVE SERVICE PROVIDER scenario these changes are only reflected in modified data requests sent by the Service Provider itself. Therefore, consumption of ACTIVE SERVICE PROVIDERS is considered to be rather stable.

EC2 *Adjustability*: An ACTIVE SERVICE REQUESTER sends both a service request to the Service Provider and a data request to the Data Provider. Hence, it is obvious that an ACTIVE SERVICE REQUESTER can adjust the data request to a specific service request. An ACTIVE SERVICE PROVIDER can also adjust the data request to a specific service request, because it receives, per definition, service requests from the Service Requester. In contrast, in ACTIVE DATA PROVIDER scenario control and data flow are always separated (i.e. neither the Service Requester nor the Service Provider is sending data requests to the Data Provider), which means the data sent by the Data Provider can not be changed in function of a specific service request.

EC3 *Coupling with Data Provider*: It is clear that an ACTIVE SERVICE PROVIDER is coupled with the external world, because it needs to send out data requests to known external parties. In contrast, in a case of an ACTIVE SERVICE REQUESTER and ACTIVE DATA PROVIDER the Service Provider simply expects that the data is provided at some point in time. In such scenarios Service Providers do not have to initiate interactions with external parties (for input data purposes). An ACTIVE SERVICE REQUESTER has a coupling with the Data Provider, but this can possibly be considered more acceptable because Service Requesters are also strongly coupled with Service Providers that need to be triggered. An ACTIVE DATA PROVIDER implies a looser coupled Service Requester and Service Provider. However, as a consequence an ACTIVE DATA PROVIDER is more coupled with the external world, because it autonomously sends out data instead of sending data upon request.

EC4 *Data provider accessibility*: Since an ACTIVE SERVICE REQUESTER needs to send a data request to the Data Provider, an ACTIVE SERVICE REQUESTER is not appropriate when only the Service Provider has access to the Data Provider.

EC5 *Confidentiality of data requirements*: As discussed in the previous evaluation criterion, an ACTIVE SERVICE REQUESTER needs to send a data request to the Data Provider. However, if the data requirements are confidential and are only known by the Service Provider itself, an ACTIVE SERVICE REQUESTER is not appropriate. As discussed in the adjustability criterion, an ACTIVE DATA PROVIDER cannot send data that is adjusted to a specific service request. As a consequence, an ACTIVE DATA PROVIDER often needs to send a larger amount of data (e.g. information regarding multiple patients). This can be favorable because in this way the *specific* data requirements are not known by the Data Provider.

EC6-8 *Data confidentiality, reusability and format*: This pattern only deals with the initiation of the data flow (see problem definition in 4.3.1). It does not describe anything about the data itself or the data transmission between services. Therefore, evaluation criteria EC6-8 are not relevant for the evaluation of this pattern.

Appendix B.2. Direct-Indirect request

EC1-2 *Robustness to change* and *adjustability*: This pattern deals with ACTIVE SERVICE PROVIDERS (see problem definition in 4.3.1). Hence, both solutions presented in this pattern, which both include ACTIVE SERVICE PROVIDERS, score equally on these evaluation criteria. As discussed in the evaluation of the DATA FLOW INITIATION pattern (see section 4.3.3), ACTIVE SERVICE PROVIDERS lead to robust and adjustable coordination scenarios.

EC3 *Coupling with Data Provider*: In the direct request scenario there is a strong coupling between the Service Provider and the Data Provider. Sending data requests to the Service Requester, as in the indirect request scenario, removes this coupling. However, note that in the indirect request scenario there is a coupling between the Service Requester and the Data Provider. Perhaps this can be considered more acceptable because Service Requesters are also strongly coupled with Service Providers that need to be triggered. In the direct request scenario only the Service Provider needs to know which service plays the role of Data Provider, while the indirect request scenario requires that the Data Provider is known by the Service Requester.

EC4 *Data provider accessibility*: The direct request scenario requires that the Data Provider is known by the Service Provider, while in the indirect request scenario only the Service Requester needs to know which service plays the role of Data Provider. Similarly, the direct request scenario requires that the Data Provider can be accessed by the Service Provider, while in the indirect request scenario only the Service Requester needs to have access to the Data Provider.

EC5 *Confidentiality of data requirements*: In the indirect request scenario the Service Requester needs to send a data request to the Data Provider. However, if the data requirements are confidential and are only known by the Service Provider itself, an active Service Provider with indirect request is not appropriate.

EC6-8 *Data confidentiality, reusability and format*: This pattern only deals with data requests (see problem definition in 4.4.1). It does not describe anything about the data itself or the data communication between services. Therefore, evaluation criteria EC6-8 are not relevant for the evaluation of this pattern.

Appendix B.3. Direct-Indirect transmission

EC1-4 *Robustness to change, adjustability, coupling with Data Provider* and *Data Provider accessibility*: When evaluating a complete coordination scenario using these evaluation criteria, this pattern has no influence on the evaluation. In other words, an evaluation of the solutions in this pattern totally depends on the specific solutions chosen in the other two patterns. For an evaluation regarding these criteria for the DATA FLOW INITIATION pattern and the DIRECT-INDIRECT REQUEST pattern, we refer to sections 4.3.3 and 4.4.3.

EC5 *Confidentiality of data requirements*: In the INDIRECT DATA TRANSMISSION scenario, all data that needs to be transmitted to the Service Provider is passed through the Service Requester. However, if the Service Provider's data requirements are confidential and can only be known by the Service Provider itself, INDIRECT DATA TRANSMISSION is not appropriate.

EC6 *Data confidentiality*: When the provided data is confidential, DIRECT DATA TRANSMISSION is the best scenario, since INDIRECT DATA TRANSMISSION implies that the data is passed through the Service Requester before it is received by the Service Provider.

EC7 *Data reusability*: INDIRECT DATA TRANSMISSION facilitates the reuse of the provided data. For example, the Service Requester only receives the specific data once, before distributing the same data to several Service Providers it interacts with.

EC8 *Data format*: INDIRECT DATA TRANSMISSION allows data transformations, since all data that need to be transmitted to the Data Provider is passed through the Service Requester. As such, the Service Requester can be responsible for data transformations. However, in a DIRECT DATA TRANSMISSION scenario the Service Requester is not involved when the data needs to be transmitted from the Data Provider to the Service Provider. As a consequence, intermediary data transformations are not possible.

Appendix C. Prolog program for completeness confirmation

```
% defining participants in a coordination scenario
participant(service_requester).
participant(service_provider).
participant(data_provider).

% defining two types of messages
message(data_request).
message(data).

% defining a message exchange between two participants
message_exchange(Participant1, Participant2, Message) :-
  participant(Participant1),
  participant(Participant2),
  Participant1\=Participant2,
  message(Message).

% C1.1: The service provider can only send data requests or receive data
possible_message_exchange(service_provider, Y, data_request) :-
  message_exchange(service_provider, Y, data_request).
possible_message_exchange(X, service_provider, data) :-
  message_exchange(X, service_provider, data).

% C1.2: The data provider can only receive data requests or send data
possible_message_exchange(data_provider, Y, data) :-
  message_exchange(data_provider, Y, data).
possible_message_exchange(X, data_provider, data_request) :-
  message_exchange(X, data_provider, data_request).

coordination_scenario(CoordinationScenario) :-
  % C1: A coordination scenario needs to be a proper interaction scenario
  findall(X, interaction_scenario(X), ListOfInteractionScenarios),
  member(CoordinationScenario, ListOfInteractionScenarios),
  % C2: the service provider must receive data from another entity
  member((_, service_provider, data), CoordinationScenario),
  % C3: the data provider must send data to an entity
  member((data_provider, _, data), CoordinationScenario),
  % C4: The resulting data flow must be complete
  complete_data_flow(CoordinationScenario),
  % requests or data can only be sent once per participant
  not(multiple_requests_or_data_sent(CoordinationScenario)),
  % requests or data can only be received once per participant
  not(multiple_requests_or_data_received(CoordinationScenario)).

interaction_scenario(MessageExchanges) :-
  setof((Participant1, Participant2, Message), possible_message_exchange(Participant1, Participant2,
       Message), L),
  sublist(MessageExchanges, L).

% C4: The data flow is complete
% Specification as the negation of not C4.1 or not C4.2
complete_data_flow(Coordination_messages) :-
  not(incomplete_data_flow(Coordination_messages)).

% C4.1: The service requester must forward any data request to the data provider.
% Specification of a coordination in which C4.1 is not true
incomplete_data_flow(Coordination_messages) :-
  member((service_provider, service_requester, data_request), Coordination_messages),
  not(member((service_requester, data_provider, data_request), Coordination_messages)).

% C4.2: An entity can only send data if this entity is the data provider or has received data
    from another entity.
% Specification of a coordination in which C4.2 is not true
incomplete_data_flow(Coordination_messages) :-
  member((Participant1, Participant2, data), Coordination_messages),
  not(member((_, Participant1, data), Coordination_messages)),
  Participant1 \= data_provider.

multiple_requests_or_data_sent(Coordination_messages) :-
  member((Participant1, Participant2a, Message), Coordination_messages),
  member((Participant1, Participant2b, Message), Coordination_messages),
  Participant2a \= Participant2b.

multiple_requests_or_data_received(Coordination_messages) :-
  member((Participant1a, Participant2, Message), Coordination_messages),
  member((Participant1b, Participant2, Message), Coordination_messages),
  Participant1a \= Participant1b.

sublist([], _).

sublist([A|B], [C|D]) :-
  (
    A = C,
    sublist(B,D)
    ;
    sublist([A|B],D)
    ).
```

Listing 1: Prolog program completeness confirmation