KATHOLIEKE UNIVERSITEIT
LEUVEN

Faculty of Business and Economics

On the concept of service coordination

Geert Monsieur, Monique Snoeck and Wilfried Lemahieu

DEPARTMENT OF DECISION SCIENCES AND INFORMATION MANAGEMENT (KBI)

KBI 1027

On the concept of service coordination

Geert Monsieur, Monique Snoeck, Wilfried Lemahieu
Faculty of Business and Economics
Katholieke Universiteit Leuven
firstname.lastname@econ.kuleuven.be

February 1, 2010

1 Introduction

In this research report we study service *coordination*, which is a crucial aspect when services are combined into service-based systems (Metzger & Pohl, 2009).

Coordination is not only in computer science an important research topic. It is also studied in disciplines such as organization theory, operations research, economics, linguistics, and psychology. Malone & Crowston (1994) studied the similarities and connections between the different flavors of coordination and created a more generic coordination theory. They define coordination as *managing dependencies* between activities. This definition is based on the intuitive idea that there is nothing to coordinate without any interdependence. In their work they state that coordination theories should try to characterize different kinds of dependencies and identify the coordination processes that can be used to manage them. Therefore, in the next section (see Section 2), we will discuss several types of dependencies that can exist when composing services to service-based systems. In the rest of this research report is targeted at two types of dependencies, namely sequence dependencies and data dependencies. Therefore, the research continues in Sections 3 and 4 with an overview of existing techniques for managing sequence and data dependencies, respectively. In each section there is also a general conclusion included.

2 Dependencies in service compositions

Multiple researchers have studied dependencies in the context of service composition (Janssen & Feenstra, 2008; Bhiri, Perrin & Godart, 2005, 2006; Yang, Papazoglou & Heuvel, 2002; Papazoglou, Delis, Bouguettaya & Haghjoo, 1997).

Bhiri et al. (2005, 2006) have used dependencies to specify how (component) services are coupled and how the behavior of some given services influences the behavior of some others. In their view on inter-service dependencies they assume that a service's behavior can be modeled using a finite state machine. In particular, they assume that a service has a minimal set of states (*initial, active, aborted, canceled, failed* and *completed*) and transitions (*activate*, *abort*, *cancel*, *fail* and *complete*). When a service is instantiated (e.g. the service received a request for executing a business task), the state of the instance is *initial*. Then this instance can be either *aborted* or *activated*. Once it is *active,* the instance can normally continue its execution or it can be *canceled* during its execution. In the first case, it can achieve its objective and successfully *complete* or it can *fail*. Additionally, the service's transactional properties can be extended by adding a *compensated* state and *compensate* transition, which can be fired when the service is in the *completed* state. Based on this behavior model Bhiri et al. (2005, 2006) consider two classes of dependencies: activation dependencies and transactional dependencies. The activation dependencies class contains two types of dependencies:

Activation dependency There is activation dependency between a service s_1 and a service s_2 if the completion of s_1 can fire the activation of s_2.

Abortion dependency There is activation dependency between a service s_1 and a service s_2 if the failure, the cancellation or the abortion of s_1 can fire the abortion of s_2.

In the transactional dependencies class a distinction is made between three types of dependencies:

Compensation dependency There is a compensation dependency from s_1 to s_2 if the failure or the compensation of s_1 can fire the compensation of s_2.

Cancellation dependency There is a cancellation dependency from s_1 to s_2 if the failure of s_1 can fire the cancellation of s_2.

Alternative dependency There is an alternative dependency from s_1 to s_2 if the failure of s_1 can fire the activation of s_2.

Janssen & Feenstra (2008) argue that the analysis of dependencies is necessary to create feasible service compositions and to identify alternative compositions. In their social-technical theory they identify two main classes of dependencies among component services: resource and link dependencies (Janssen & Feenstra, 2008).

Resource dependency There is a resource dependency between two services if both services use the same resource, which sets constraints on the execution order (i.e. a service should be consumed before another service can start).

Link dependency There is a link dependency between two services if one service depends in some way on the output of the other service (e.g. a service cannot continue its execution until another service has provided certain data)

Yang et al. (2002) and Papazoglou et al. (1997) make a distinction between two types of dependencies that can occur among service components:

Sequence dependency There is a sequence dependency (also referred to as a *commit dependency*) between a service s_1 and a service s_2 if the start *or* continuation of the execution of service s_2 depends on the completion of the execution of s_1.

Data dependency There is a data dependency between a service s_1 and a service s_2 if the start *or* the continuation of the execution of service s_2 depends on data that is provided by s_1.

In our research we use the two dependencies as proposed by Yang et al. (2002), because these two types are more general than the dependencies proposed by Bhiri et al. (2005, 2006) and Janssen & Feenstra (2008). The dependencies presented by Bhiri et al. (2005, 2006) are mostly sequence dependencies, while Janssen & Feenstra (2008) listed both sequence and data dependencies.

Based on the distinction between sequence and data dependencies, we can identify two main coordination challenges that one faces when composing service-based systems. The first challenge is the management of sequence dependencies. This challenge consists of consuming all component services in the right order (i.e. according to sequence dependencies as possibly specified in a business process). For example, an order fulfillment process could have a sequence dependency between a payment service and a shipping service, because this business process specifies that the order can only be shipped after the payment is arranged. The second challenge is the management of data dependencies, which is about providing a service with all the data it needs. Consider for instance a data dependency between a shipping service and a customer relationship service, because the shipping service can only ship an order if it received the customer's shipping address, which can be provided by the customer relationship service. Obviously, each complete coordination scenario (for an order fulfillment process) needs to take into account both sequence and data dependencies.

In their Extended SOA Papazoglou (2005) and Papazoglou & 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 and managing the data flow, exactly is what managing sequence and data dependencies is about.

Alonso, Casati, Kuno & Machiraju (2004) consider six different dimensions of a service composition model. Among these are an *orchestration model*, which Alonso et al. (2004) define as a model that specifies the order in which services are to be invoked, and a *data and data access model*, which they define as a model that describes how data is specified and how it is exchanged between components. Hence, in an orchestration model it is specified how sequence dependencies are managed, while a data and data access model describes how data dependencies are dealt with.

3 Managing sequence dependencies

3.1 Introduction

Many developers use the terms *orchestration* and *choreography* to describe the business interaction protocols that control the execution of services (Papazoglou, Traverso, Dustdar & Leymann, 2007). Since managing sequence dependencies is about controlling the execution of services and designing the appropriate service interactions for accomplishing this, orchestration and choreography must be relevant terms when designing coordination logic for managing sequence dependencies.

However, orchestration and choreography are not always defined in the same way. Furthermore, some researchers introduce centralized or decentralized variants such as *centralized orchestration*, *decentralized orchestration*, *centralized choreography* and *decentralized choreography*. In addition, it is not always clear how these terms are related to terms such as *centralized coordination*, *decentralized coordination* and *peer-to-peer coordination*. In the same way, it is not always easy to understand the relations to concepts such as a *coordinator*, an *orchestrator* or a *choreographer*. To make things even more complex, there exist several synonyms or overlapping concepts such as *behavioral interface*, an *abstract process* in BPEL (OASIS, 2007) and an *executable process* in BPEL (OASIS, 2007).

In order to understand terms like orchestration and choreography (and related concepts) it is beneficial to have a set of basic concepts that are related to service composition in general, regardless of whether an orchestration- or a choreography-oriented description is used. In Subsection 3.2 we discuss such basic concepts. Subsequently, we use these concepts to discuss two groups of orchestration and choreography definitions that can be found in the literature (see Subsections 3.3 and 3.4).

3.2 A meta-model for service composition

Figure 1 shows our meta-model for service composition design, illustrating common concepts in service composition design and their relationships using the UML class diagram notation (OMG, 2010). It extends the meta-model proposed by Benatallah, Dijkman, Dumas & Maamar (2005) by adding business process-related concepts such as business process, business task, business request and business event.

A service *composition* supporting a *business process* consists of a number of (component) *services* that are provided by *service providers*. The same service can be provided more than once by different service providers (e.g. a flight booking service can be provided by different airlines). A service has several (internal) *actions* and is associated to several *message events* that are part of an *interaction* with other services. The interactions are message-based because this is the basic mechanism for interaction used in the mainstream service description languages (e.g. WSDL (W3C, 2001, 2007)). Each interaction consists of a *catching message event* (receive) and a *throwing message event* (send). A message event is associated with a *message type*, which is either a *business request* or a *business event*. A *business request* - also referred to as a *service request* - means that the service sends (in case of a throwing message event) or receives (in case of a catching message event) a request for executing a certain business task in the business process. Business requests typically are defined in a *contract* between two parties in which is stated what the requester can expected from the party that receives the request (Weigand & Van den Heuvel, 2002). A *business event* means that the service sends or receives a event notification that describes a business task-related event (e.g. the end of a certain business task) (Hens, Snoeck, Poels & De Backer, 2009). *Relations* relate actions and interactions to each other via message events. The interactions (including the message events) form the coordination logic in a service composition.

In this meta-model abstraction is made of the sequence constraints between interactions. Furthermore, the meta-model does not take account the transactional aspect of coordinating business processes and tasks.

3.3 Orchestration and choreography as composition viewpoints

A first group of researchers consider orchestration and choreography simply as different *viewpoints* in a service composition. The most detailed definitions in this category are proposed by Barros, Dumas & Oaks (2006). Therefore, we will summarize their definitions below and indicate the relationship with the meta-model presented in Subsection 3.2. Subsequently, we highlight similar definitions by other researchers.

Barros et al. (2006) define a choreography (model) as follows:

- *"A description of a collaboration between a collection of services to achieve*

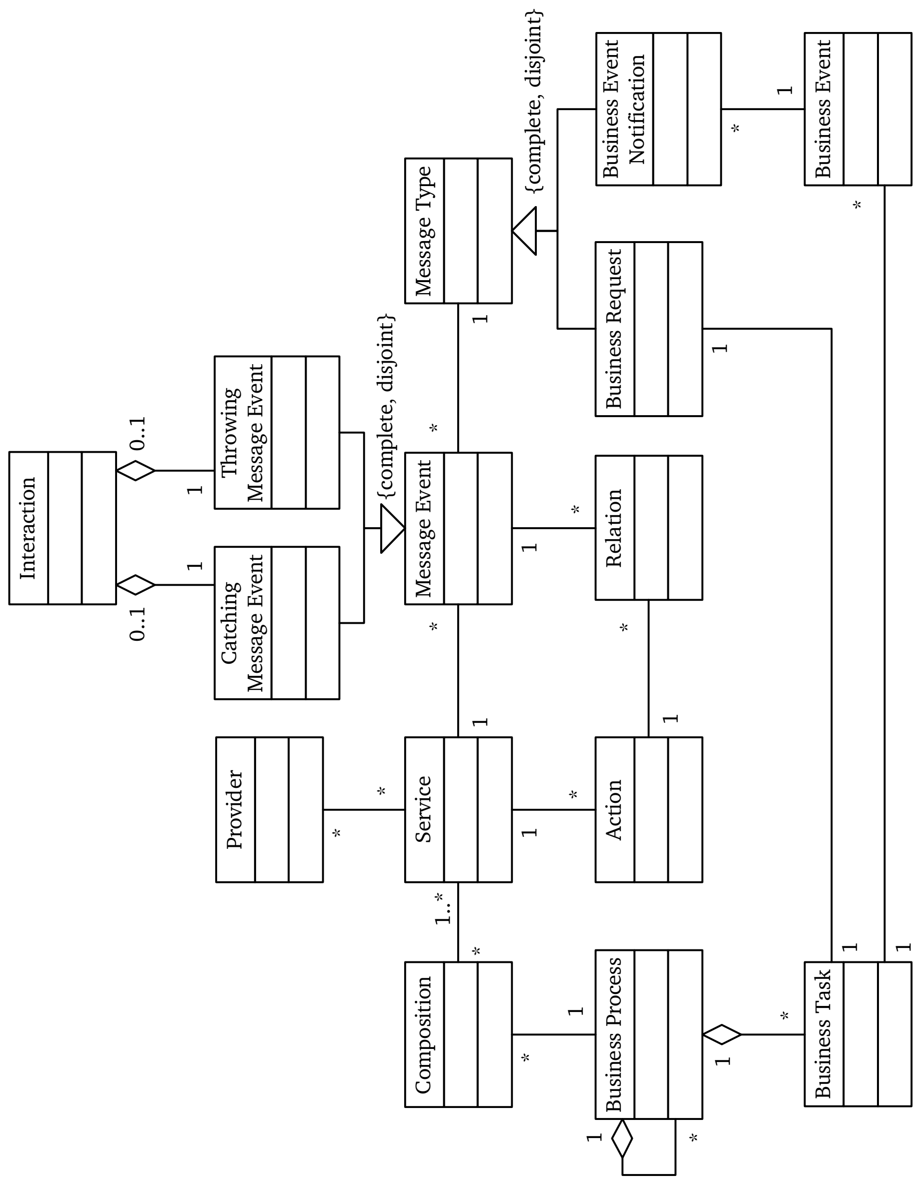


Figure 1: Basic concepts in service compositions

a common goal.

- *"It captures the interactions in which the participating services engage to achieve this goal and the dependencies between these interactions, including the causal and/or control-flow dependencies (i.e. that a given interaction must occur before another one, or that an interaction causes another one)"*
- *"The interactions are captured from a global perspective meaning that all participating services are treated equally. In other words, a choreography encompasses all interactions between the participating services that are relevant with respect to the choreography's goal."*
- *"It does not describe any internal action of a participating service that does not directly result in an externally visible effect, such as an internal computation or data transformation."*

Hence, choreography is about the design of all interactions, including the message events, in a service composition. In a service composition component services are collaborating together to achieve a common goal, which is the realization of a new composite service. A choreography does not describe the relationship between these interactions and the internal actions of the component services.

Barros et al. (2006) define an orchestration (model) as *"a description of the communication actions and the internal actions in which a service engages. Internal actions include data transformations and invocations to internal software modules."* Thus, orchestration is about designing all (internal) actions and related message events and interactions in which a *particular* component service is involved.

The above definitions of orchestration and choreography fit well in several business-to-business service composition methodologies, in which both orchestration and choreography (as viewpoints) need to be combined in order to build a service-based application that supports the inter-enterprise business process. The main idea in these methodologies (Papazoglou et al., 2007; Papazoglou & Van den Heuvel, 2007; Peltz, 2003) is that companies involved in the business process first agree upon on the way they collaborate by designing and specifying a choreography. Subsequently, each participating company can develop an orchestration that fits into the choreography. In practice companies first design a choreography (e.g. in a WS-CDL (W3C, 2005) or BPEL4Chor (Decker, Kopp, Leymann & Weske, 2009) representation) and then generate a sort of orchestration skeleton for each company (e.g. in an abstract BPEL process (OASIS, 2007)). Barros et al. (2006) refer to this orchestration skeleton as a behavioral interface:

- *"A behavioral interface captures the behavioral aspects of the interactions in which one particular service can engage to achieve a goal."*

- *”It focuses on the perspective of one single party (in a choreography). As a result, a behavioral interface does not capture ’complete interactions’ since interactions necessarily involve two parties.”*

- *”Basically, it consists of communication actions performed by one participant”*

- *”A behavioral interfaces does not describe internal tasks such as internal data transformations*.

Note that these definitions imply that in each service composition there is only one choreography viewpoint and several orchestration viewpoints.

3.4 Orchestration and choreography as composition styles

A second group of researchers describe orchestration and choreography as composition *styles*, which are two generic ways that illustrate how component services in a service composition interact. This means that orchestration and choreography can be considered as two *templates* for coordination scenarios that manage sequence dependencies.

Several researches define orchestration as a service composition in which a central coordinator - sometimes also referred to as the orchestrator (Busi, Gorrieri, Guidi, Lucchi & Zavattaro, 2005; Tabatabaei, Kadir & Ibrahim, 2008; ter Beek, Bucchiarone & Gnesi, 2006), the controller (Barker, Weissman & Hemert, 2009) or the choreographer (Mitra, Kumar & Basu, 2008) - is responsible for coordinating the business process execution by invoking all component services in the right order (Busi et al., 2005; Barker et al., 2009; Tabatabaei et al., 2008; Malinova & Gocheva-Ilieva, 2008; ter Beek et al., 2006; Bellini, Prado & Zaina, 2010). This coordinator can be either one of the component services or another service (e.g. a BPEL engine) (Malinova & Gocheva-Ilieva, 2008). Pedraza & Estublier (2009) describe orchestration as a service composition in which *”a single engine on a single machine forms the heart of the (service-based) system, with all communication going to and coming from that machine”*. Similarly, Pessoa, Silva, Sinderen, Quartel & Pires (2008) state that all the interactions in an orchestration pass through the coordinator.

In summary, this means that orchestration can be considered as a service composition style in which one service (the coordinator) interacts with all component services. There are no (explicit) interactions between component services; each interaction in the orchestration is between the coordinator and a component service. Typically, only the coordinator has knowledge of the business process and sends business requests to the component services. The component services mostly send business events to the coordinator. In the literature, orchestration defined in this way is also referred to as *centralized coordination* (Pessoa et al., 2008; Benatallah, Sheng & Dumas, 2003), *centralized orchestration* (Binder, Constantinescu & Faltings, 2006) or *centralized choreography* (Mitra et al., 2008). This definition can also be explained by means of the orchestra metaphor. As in a real-life orchestra, one service is playing the role of the conductor and coordinates all component services (Lin & Chang, 2005).

In line with orchestration as a composition style, choreography is often described as a service composition in which there is *no* central coordinator. In a choreography the component services are rather collaborating together. The business process is executed and coordinated by several peer-to-peer interactions among the collaborating services (Busi et al., 2005; Barker et al., 2009; Tabatabaei et al., 2008; Malinova & Gocheva-Ilieva, 2008; ter Beek et al., 2006; Bellini et al., 2010). The component services directly communicate with each other, and not through a central coordinator (Pedraza & Estublier, 2009; Pessoa et al., 2008). According to Barker et al. (2009) and Malinova & Gocheva-Ilieva (2008) all participants in a choreography need to be aware of the business process, operations to execute, messages to exchange, and the timing of message exchanges. In contrast, Pedraza & Estublier (2009) state that each participant is responsible for routing messages to the 'next' (Web) service(s) *without* any global view such as the business process. In summary, choreography can be considered as a service composition style in which the component services interact with each other and there is no central coordinator that exclusively interacts with component services. Typically, several component services have knowledge about (part of) the business process and send business requests to each other; component services mostly also send business events to other component services. In the literature, choreography defined in this way is also referred to as *peer-to-peer coordination* (Pessoa et al., 2008; Benatallah et al., 2003) or *decentralized orchestration* (Binder et al., 2006). This definition can also be explained by means of the dancers metaphor. As a group dancers do in real life, each service knows exactly when to execute and with whom to interact (Lin & Chang, 2005).

3.5 Conclusion

As discussed in the introduction of this section, orchestration and choreography are important terms when designing coordination logic that deals with sequence dependencies. Based on our literature overview (see Subsections 3.3 and 3.4) we can conclude that orchestration and choreography are relevant terms in two ways. On the one hand, orchestration and choreography can be considered as viewpoints in a service composition. In particular, one can say that orchestration and choreography provide local and global views on a coordination scenario. On the other hand, orchestration and choreography can refer to two composition styles. In the context of service coordination, this perspective is probably the most relevant, because in a way these composition styles refer to different kinds of coordination scenarios.

We believe both composition viewpoints and styles are useful concepts when designing service interactions. However, as shown in the previous subsections the terms orchestration and choreography are used for both purposes, which can potentially lead to confusion and inconsistencies.When referring to the composition styles, we advice to use the terms *centralized* and *decentralized coordination*. Terms such as *centralized orchestration* or *decentralized orchestration* are very confusing when orchestration is considered as a viewpoint, because then orchestration is, per definition, always executed by one service. *Centralized choreography* or *decentralized choreography* are perhaps easier to understand, because then the adjectives tell something about the interactions. However, these terms, currently, are not so frequently used in the literature.

Note that it is relatively easy to understand that people confuse composition viewpoints with composition styles. Indeed, from the viewpoint perspective orchestrations are always executed by one service. However, it is important to realize that this does not necessarily imply that the service executing the orchestration have to coordinate the complete business process. Similarly, it is true that in a choreography (as viewpoint) there are several services interacting with each other, but this does not imply a centralized or decentralized coordination. Indeed, also in a centralized coordination one can consider the choreography viewpoint.

As mentioned earlier, in service coordination is mainly related to composition styles, because centralized or decentralized coordination describes a sort of coordination style. Nevertheless, orchestration and choreography as composition styles does not explicitly describe the types of messages that are exchanged between services (i.e. business requests or business event notifications). As we have described in Subsection 3.4, centralized coordination (or orchestration as composition style) typically implies that one coordinating service has business process knowledge and sends *business requests* to the component services. However, one could also think of a coordination scenario in which a central business process-agnostic service (e.g. an event broker) does not send business requests but only forwards *business event notifications* to interested services. Similarly, a choreography as composition style does not describe the nature of inter-service communication. On the one hand, one can easily imagine scenarios in which component services collaborate together and send business requests to each other. On the other hand, however, services can also collaborate together by only sharing business events. For example, in a group of dancers typically dancers do not tell each other what and when to do something (i.e. they do not send business requests to each other). Dancers rather react on movements etc. from others to know when to do something. In summary, we can conclude that the common distinction between orchestration and choreography does not describe the complete set of possible coordination scenarios. However, currently, there are no concrete guidelines available for building such scenarios and choosing an appropriate solution.

4 Managing data dependencies

In this section we will discuss existing solutions for managing data dependencies in service compositions. Before discussing these studies in detail, we first describe a short running example that we use for illustrating the results of these studies.

4.1 Short running example

We use a simple (fictive) example of a service composition in hospitals as running example. We deliberately chose a non-automated example so as not to clutter the discussion with (low-level) implementation issues, but the conclusions presented in this section are equally applicable to software services.

Consider a nurse who wants to treat a patient's high fever using a febrifuge. Getting the right febrifuge requires consumption of several services. In particular, the nurse should request a febrifuge from the pharmacist, who of course also provides several services to the hospital staff. 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. However, aspirin has the unpleasant side effect that it can cause stomach bleeding in certain circumstances. Therefore, the pharmacist needs patient-specific information concerning the risk for stomach bleeding, before he or she can deliver an appropriate febrifuge. The risk for stomach bleeding is only known by the patient's doctor. This means that the doctor provides a second service that needs to be consumed in order to support the task of choosing a febrifuge. Hence, this implies that there is a data dependency in this service composition between the pharmacist and the doctor. In this case, service coordination means providing the pharmacist with the data that is held by the doctor.

Even though this is a rather small example of data dependencies in a service composition, many coordination scenarios are possible. In figures 2(a) and 2(b) two ways of managing the data dependency between the pharmacist and the doctor are shown.

4.2 Existing techniques for dealing with data dependencies

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

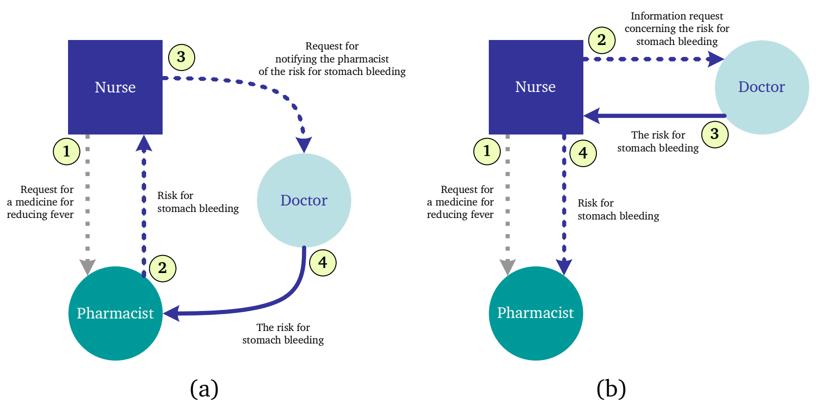
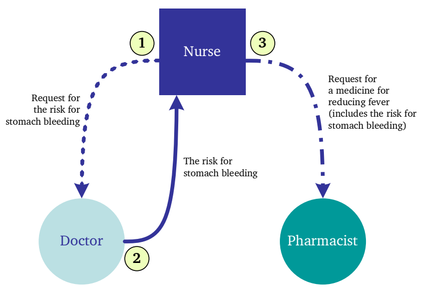


Figure 2: Two ways of coordinating the pharmacist and doctor

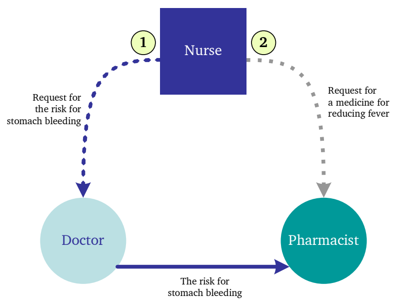
necessary for sending data from one service that can provider certain data to another service that needs that data (Barker, Weissman & Van Hemertb, 2008b; Yang, 2003; Weber, Schuler, Neukomm, Schuldt & Schek, 2003; Charfi & Mezini, 2007). A data flow thus specifies how data dependencies are managed. Therefore, we will focus on studies that contain approaches to find and use alternative data flows in a service composition. We illustrate the results of these studies by means of the running example presented in the Subsection 4.1. Subsequently, 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 3(a) and 3(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 3(a)) is used to indicate that the data is included in the invocation message. While in figure 3(a) all data passes via the nurse (central data flow), the data flow in figure 3(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.b (2008b) and Barker, Weissman & Van Hemerta (2008a) have presented a Web services based architecture that allows centralizing component invocations (centralized control flow) and decentralizing data flows (similar to figure 3(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


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

(a) Central data flow


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

(b) Decentral data flow

Figure 3: Two possible data flows for hospital example

vicinity of Web services. These proxies realize the more efficient data flow between component services.

Liu, Law & Wiederholda (2002a); Liu, Law & Wiederholdb (2002b) 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 3(b) outperforms 3(a)). Subsequently, they developed a Flow-based Infrastructure for Composing Autonomous Services (FICAS) (Liu et al.a, 2002a; Liu et al.b, 2002b). 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, Padhye, Prasad & Navathe (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.

4.3 Conclusion

We can conclude that several approaches exists that cater for alternative data flows. Many studies propose architectural infrastructures for such data flows. These infrastructures 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 styles nor do they analyze the advantages and disadvantages of alternatives. As a consequence, they fail to exhaustively identify all 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 3(a) can be transformed into a scenario such as shown in figure 3(b). However, one can easily see that there are more possibilities. For example, the scenario represented in figure 4 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 shown in the subsection 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 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

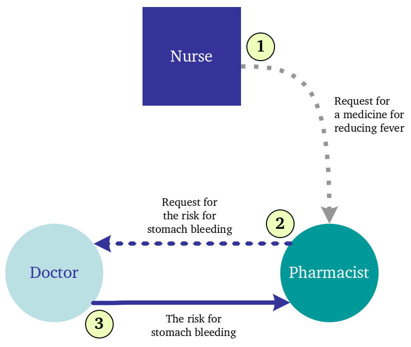

Nurse
1
Request for a medicine for reducing fever
Request for the risk for stomach bleeding
2
Doctor
Pharmacist
3
The risk for stomach bleeding

Figure 4: An alternative data flow for the scenarios represented in figures 3(a) and 3(b)

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 4 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.

References

Alonso, G., Casati, F., Kuno, H. & Machiraju, V. (2004). *Web services: concepts, architectures and applications*. Data-Centric Systems and Applications. New York, NY, USA: Springer-Verlag Berlin Heidelberg.

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* (p. 121.1). Washington, DC, USA: IEEE Computer Society.

Barker, A., Weissman, J. B. & Hemert, J. I. (2009). The circulate architecture: avoiding workflow bottlenecks caused by centralised orchestration. *Cluster Computing*, *12(2)*, 221–235.

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)* (pp. 55–64). New York, NY, USA: ACM.

Barker, A., Weissman, J. B. & Van Hemert, J. (2008b). Orchestrating datacentric workflows. In *Proceedings of the 2008 Eighth IEEE International*

Symposium on Cluster Computing and the Grid (CCGRID 2008) (pp. 210–217). Washington, DC, USA: IEEE Computer Society.

Barros, A., Dumas, M. & Oaks, P. (2006). Standards for web service choreography and orchestration: Status and perspectives. In C. Bussler & A. Haller (Eds.), *Business Process Management Workshops*, Volume 3812 of *Lecture Notes in Computer Science* (pp. 61–74). Springer-Verlag Berlin Heidelberg.

ter Beek, M., Bucchiarone, A. & Gnesi, S. (2006). A survey on service composition approaches: From industrial standards to formal methods. Technical Report 2006-TR-15, Consiglio Nazionale delle Ricerche, Pisa, Italy.

Bellini, A., Prado, A. F. d. & Zaina, L. A. M. (2010). Top-down approach for web services development. In *Proceedings of the 2010 Fifth International Conference on Internet and Web Applications and Services (ICIW 2010)* (pp. 426–431). Washington, DC, USA: IEEE Computer Society.

Benatallah, B., Dijkman, R., Dumas, M. & Maamar, Z. (2005). Service Composition: Concepts, Techniques, Tools and Trends. In *Service-Oriented Software System Engineering: Challenges and Practices* chapter 3, (pp. 48–66). IGI Publishing.

Benatallah, B., Sheng, Q. & Dumas, M. (2003). The Self-Serv Environment for Web Services Composition. *IEEE Internet Computing, 7(1)*, 40–48.

Bhiri, S., Perrin, O. & Godart, C. (2005). Ensuring required failure atomicity of composite web services. In *Proceedings of the 14th international conference on World Wide Web (WWW 2005)* (pp. 138–147). New York, NY, USA: ACM.

Bhiri, S., Perrin, O. & Godart, C. (2006). Extending workflow patterns with transactional dependencies to define reliable composite web services. In *Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW 2006)* (p. 145). Washington, DC, USA: IEEE Computer Society.

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)* (pp. 869–876). Washington, DC, USA: IEEE Computer Society.

Busi, N., Gorrieri, R., Guidi, C., Lucchi, R. & Zavattaro, G. (2005). Towards a formal framework for choreography. In *Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE 2005)* (pp. 107–112). Washington, DC, USA: IEEE Computer Society.

Charfi, A. & Mezini, M. (2007). AO4BPEL: An Aspect-oriented Extension to BPEL. *World Wide Web, 10(3)*, 309–344.

Decker, G., Kopp, O., Leymann, F. & Weske, M. (2009). Interacting services: From specification to execution. *Data & Knowledge Engineering*, *68(10)*, 946–972.

Hens, P., Snoeck, M., Poels, G. & De Backer, M. (2009). The use of the concept of event in enterprise ontologies and requirements engineering literature. Technical Report KBI 0909, Faculty of Business and Economics, Katholieke Universiteit Leuven, Leuven, Belgium.

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)* (pp. 323–330). New York, NY, USA: ACM.

Lin, F.-r. & Chang, H.-c. (2005). The development and evaluation of exception handling mechanisms for order fulfillment process based on bpel4ws. In *Proceedings of the 7th international conference on Electronic commerce (ICEC 2005)* (pp. 478–484). New York, NY, USA: ACM.

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)* (pp. 158–165). New York, NY, USA: ACM.

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)*. Louisville, Kentucky USA: ISCA.

Malinova, A. & Gocheva-Ilieva, S. (2008). Using the Business Process Execution Language for Managing Scientific Processes. *International Journal Information Technologies and Knowledge*, *2(3)*, 257–261.

Malone, T. & Crowston, K. (1994). The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR)*, *26(1)*, 119.

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

Mitra, S., Kumar, R. & Basu, S. (2008). Optimum decentralized choreography for web services composition. In *Proceedings of the 2008 IEEE International Conference on Services Computing (SCC 2008)* (pp. 395–402). Washington, DC, USA: IEEE Computer Society.

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

OMG (2010). OMG Unified Modeling Language^{TM}(OMG UML), Superstructure Version 2.3. OMG Document (formal/2010-05-05).

Papazoglou, M. (2005). Extending the service-oriented architecture. *Business Integration Journal*, *7(1)*, 18–21.

Papazoglou, M., Delis, A., Bouguettaya, A. & Haghjoo, M. (1997). Class library support for workflow environments and applications. *IEEE Transactions on Computers*, *46(6)*, 673–686.

Papazoglou, M. & Heuvel, W. (2007). Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal - The International Journal on Very Large Data Bases, 16(3)*, 415.

Papazoglou, M., Traverso, P., Dustdar, S. & Leymann, F. (2007). Service-Oriented Computing: State of the Art and Research Challenges. *Computer* (pp. 38–45).

Papazoglou, M. P. & Van den Heuvel, W.-J. (2007). Business process development life cycle methodology. *Communications of the ACM, 50(10)*, 79–85.

Pedraza, G. & Estublier, J. (2009). Distributed Orchestration Versus Choreography: The FOCAS Approach. In *Proceedings of the International Conference on Software Process (ICSP 2009)* (pp. 75–86). Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg.

Peltz, C. (2003). Web services orchestration and choreography. *Computer, 36(10)*, 46–52.

Pessoa, R. M., Silva, E., Sinderen, M. v., Quartel, D. A. C. & Pires, L. F. (2008). Enterprise interoperability with soa: a survey of service composition approaches. In *Proceedings of the 2008 12th Enterprise Distributed Object Computing Conference Workshops (EDOCW)* (pp. 238–251). Washington, DC, USA: IEEE Computer Society.

Tabatabaei, S., Kadir, W. & Ibrahim, S. (2008). Web Service Composition Approaches to Support Dynamic E-Business Systems. *Communications of the IBIMA, 2*, 115–121.

W3C (2001). Web Services Description Language (WSDL) Version 1.1. W3C Note.

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

W3C (2007). Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C 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)* (pp. 1081–1084). VLDB Endowment.

Weigand, H. & Van den Heuvel, W.-J. (2002). Cross-organizational workflow integration using contracts. *Decision Support Systems, 33(3)*, 247–265.

Yang, J. (2003). Web service componentization. *Communications of the ACM, 46(10)*, 35–40.

Yang, J., Papazoglou, M. P. & Heuvel, W.-J. V. d. (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)* (p. 125). Washington, DC, USA: IEEE Computer Society.