An overview on protocol adaptors for service component integration

An Overview on Protocol Adaptors for Service Component Integration

R. Seguel[1], R. Eshuis, P. Grefen

Information Systems Group, School of Industrial Engineering,
Eindhoven University of Technology, The Netherlands

Abstract

The Service Oriented Architecture (SOA) paradigm is intended for reducing development costs and software reuse by enabling automated service compositions for cross-organizational processes. However, if two service components need to interact, they can get stuck if their protocols are incompatible. So, an adaptor should be generated to reconcile mismatches. Existing approaches of service adaptation are manual or semi-automatic rather than fully-automated. An overview of these different approaches is presented, revealing their similarities and differences.

Key words: SOA, Adaptor, Service Composition, Service Adaptation, Process Integration.

1. Introduction

Service-orientation is moving from systems integration to more business-centric systems, leading to companies building complex architectures [53]. These architectures need industry standards for enabling a flexible and scalable design, communication among heterogeneous platforms, and best practices and methodologies to govern their lifecycle [83], according to changes in business needs and/or law regulations [24]. Service invocations and interactions become more complex due to stateful services rather than stateless ones. Service Oriented Architecture (SOA) [82] enables an increasingly integration among enterprise systems through automated service composition and service coordination. That automation actually can partially be done with existing technologies. Full automation is not possible yet because of integration issues among stateful services.

These integration issues are related to interface [70] and behavioral mismatches [11, 16, 30, 59, 62, 71] between service components. Two service components cannot interact if their interface declarations mismatch. That is, the messages exchanged have different formats and specifications. Interface level mismatches can be resolved using schema mapping and transformation tools. Two service components have a behavioral mismatch if their protocols get stuck during the interaction. Service adaptation examines the service protocols compatibility between two service components. Behavioral mismatches are identified in the protocols by analyzing the

Email address: r.e.seguel@tue.nl (R. Seguel)
[1]Tel. +31 040 2472290, P.O. Box 513, 5600MB Eindhoven, The Netherlands.

ordering constrains of the messages exchanged and the existence of deadlocks. Then, a protocol adaptor is defined to reconcile the differences of two incompatible protocols. Actually, there are manual and semi-automated approaches for generating protocol adaptors. In this paper, we give an overview of these different adaptation approaches, highlighting their similarities and differences.

There are a few other papers that survey adaptation approaches. Canal et al. [23] give a detailed description of software adaptation in the context of component-based software engineering, while we focus on Web services. Dumas et al. [31] give a brief overview on compatibility and adaptation for service protocols. In this paper we also focus on service protocols, but we give a broader and more detailed overview of the different semi-automated adaptation approaches than Dumas et al. [31].

The remaining sections are organized as follows. Section 2 describes the context of adaptation in service-orientation. It outlines the integration issues in the emerging service oriented technologies. Service composition combines services according to a business protocol. Two service component cannot interact properly if their protocols mismatch. Then an adaptor can be used to reconcile the differences. Also, in this section we show current service composition technologies that are relevant for the context of this paper. Afterwards, in Section 3, we show a motivating example describing a case where two service components cannot interact because they get stuck. We focus on protocol mismatches rather than interface mismatches. We describe what is protocol compatibility and which communication semantics is used for analyzing protocols. In Section 4, we use these arguments and the example to manually build an adaptor for reconciling these protocol mismatches. However, we show there are more adaptors for the same example. Also, we show these adaptors are deduced using different assumptions where service components interact in different ways allowed by the protocols and the environment. Since a manual mechanism is error-prone, we show the current semi-automated approaches for generating a protocol adaptor. We detail the main similarities and differences between the most important approaches, and show a summarizing table. Also, we show that some of the manually generated adaptors can be developed too using the semi-automated approaches. Finally, in Section 5 we explain the conclusions and further work in the area.

2. Service component integration issues

In this section, we describe the main integration issues of service components and the service oriented technologies involved in the service composition setting. In this setting, we show that the integration issues in service oriented systems are similar to those present in EAI systems.

2.1. Integration issues on service-based systems

Companies have an increasing need for integrating legacy systems to access their information sources easier. Workflow Management Systems (WfMS) [57, 92] define and control the processes in a company. These systems control the information flows between people and applications. An increasingly number of heterogeneous applications into a company leads to complex information flows, requiring application integration. That application integration allows companies to define and compose new process interactions. The integration can be done using Enterprise Integration Application (EAI) systems [6, 47, 86]. In this case, companies develop application-specific adaptors (wrappers) allowing the integration of all sorts of applications [3]. However, these adaptors should be developed for each type of application. So, the definition

and composition of new processes becomes hard to maintain, error-prone and without allowing to reuse software pieces. Moreover, even though Workflow Management Systems (WfMS) and Enterprise Application Integration (EAI) systems are very flexible and generic [57], these are limited to LANs rather than broader networks [3]. That limitation also hinders the integration of processes between companies.

However, emerging service oriented technologies provide standards to compose and integrate intra- and cross-organizational process interactions [41, 42] through Web Services [95], making interoperability easier. Web services [15, 39] are a prime example of electronic services. A service can be very simple as one which converts an amount of money to another, or very complex as one that invokes complex business applications [45]. In this setting, business processes invoke Web service operations rather than conventional applications [3].

Companies build Service Oriented Architectures (SOAs) [15, 76, 89] using Web service technologies [4, 9, 27, 26, 28, 46] for enabling interoperability and systems integration [66], reducing development costs and allowing software reuse. SOA involves service components of different business partners, so process descriptions are needed to adjust interactions among different partners [33]. A conversation is referred to the sequences of operations (message events) that occur between two services as part of a Web service invocation [22, 40]. Web service transactions [58, 93, 94, 97] are an important aspect of loosely coupled business interactions between independent parties [44].

A service composition [40, 56, 75] is a set of interacting service components communicating with each other via either synchronous or asynchronous messaging. Service components [34] communicate according to a business protocol that specifies the order which the messages are exchanged. Two service components can get stuck during a dynamic interaction due to behavioral constraints in their protocols. Therefore, there is no properly terminating service composition in that case. These intra- or cross-organizational process interactions cannot be executed. Moreover, the messages exchanged must satisfy the syntactic WSDL [27, 26] declaration of both services in order to avoid incompatible types. Transformation definition and schema mapping tools are available for interface adaptation [32]. So, in this paper we focus in resolving behavioral mismatches produced dynamically on Web service conversations rather than static checking of interface syntactics [16].

If two service components have protocol mismatches, they cannot interact properly. A protocol adaptor can be developed to reconcile such differences. Adaptors can be used to ensure backwards compatibility of new service versions and compatibility with different client classes [20]. However, service adaptation is in its early stages since current approaches are only partial solutions [20].

The integration of business processes across the boundaries of organizations involves aspects like: control and data flow integration, visibility of process details, transactional behavior of processes, and quality of service aspects. For the context of this paper, a non-black box interface for service components is assumed since input/output and protocol information are needed in stateful services [35, 44, 84]. In this area, the CrossFlow [43] project defines support for cross-organizational workflow management in dynamic virtual enterprises. CrossFlow uses an outsourcing paradigm including contract establishment and workflow enactment for executing services. Although CrossFlow supports automated, dynamic setup of cross-organizational processes, the approach relies on an asymmetric service outsourcing paradigm. In that paradigm, an organization outsources a predefined part of its business process to a service provider. So, this is too limited for a multi-party and peer-to-peer situation [45]. The CrossWork [29] project overcomes these limitations. CrossWork has designed concepts, an architecture and technology

for supporting semi-automated business process management in Network of Automotive Excellence (NoAEs). It allows dynamic workflow formation and enactment [44]. Also, this enables tight collaboration and strong synergies between different autonomous organizations [45, 74]. The CrossWork architecture defines an environment for a cross-organizational business process setup, verification, and enactment that integrates legacy systems [74]. Recently, a reference architecture for e-contracting has been defined in [5], which can be used as a standardization model that facilitates systems integration.

In sum, service-based systems have similar integration issues as EAI systems: nowadays, companies need to develop adaptors to reconcile differences between two service components (service-based systems) rather than applications (EAI systems). These adaptors should be developed on-the-fly rather than statically in order to facilitate maintenance, reduce errors and development costs, and allow software reuse.

In the next subsection, we present the technologies involved in service composition for the context of this paper.

2.2. Service composition technologies

The full potential of stateful services is achieved by using WS-BPEL [4] as the standard process integration model for complex service interactions. WS-BPEL is a standard language for business protocols [3, 33, 52, 90]. Also, it is used to define the internal implementation of composite services (executable processes) and the external behavior of services (abstract processes). WS-BPEL uses WSDL [33, 53] and provides activities for the specification of a business process. These activities have a predefined behavior divided in basic and structured [4]. Basic activities specify the interaction between the process and its partner. A structured activity specifies ordering constraints on its child activities, and these are either other structured activities or basic activities [35].

WS-BPEL is an orchestration language [36, 88]. *Service Orchestration* refers to the execution of specific business processes describing the interaction from the perspective of one component. That is centralized and one service acts as a controlling hub in relation with the other. In contrast, *Service Choreography* refers to externally observable interactions described between services without a controlling hub [59, 88]. For the latter, the Web Services Choreography Description Language (WS-CDL) has been defined by W3C [51].

Service Component Architecture (SCA) [9] is a new effort of industry leading companies. This defines a standard model for building applications and systems using a Service-Oriented Architecture. SCA extends and complements prior approaches for implementing services. That is built on top of open standards such as Web Services [4, 26, 27, 28, 46]. Also, it provides a series of services which are assembled together to create solutions that serve for a particular business need [9, 89, 63]. SCA and WS-BPEL are complementary with one another [9]. SCA provides a compositional view of interconnection among service components and WS-BPEL provides a coordination and composition view of business protocols. In the context of this paper, we focus on service composition of business protocols and the technologies involved there such as WS-BPEL.

Service composition distinguishes six dimensions of a composition model [3]: a component model, an orchestration model, data and a data access model, a service selection model, transactions, and a exception handling mechanism. Self-Serv [12] is an example of a scalable framework for supporting the model-driven development and decentralized execution of composite services. In that framework, services are distinguished on three types: elementary, composite

and communities. Moreover, it uses statecharts as an orchestration model. Also, it provides peer-to-peer interaction schema that ensures control and data flow dependencies between composites services. Another approach based on semi-automated service composition is defined for composing a given set of services into a structured process model [36]. This approach defines dependency graphs between services according to the input and output messages of each service. After that, these abstract dependencies are typed with concrete branching types like AND and XOR. Finally, the concrete dependencies are used to compose the services into a structured process model [36]. Fully automatic approaches mostly come from the formal reasoning field that requires the services are specified formally with pre- and post-conditions [36].

On the other hand, SOA provides a framework for integrated services between companies without using semantics interoperability definitions [25, 87]. However, two initiatives provide explicit representation of integration concepts: the DARPA Agent Markup Language for Web Services (DAML-S) based on the ontology language OWL-S [65] and the Web Service Modeling Ontology (WSMO) [55] based on the Web Service Modeling Language (WSML) [21]. Also, a reference architecture has been defined for a Semantic Business Process Management System (SBPMS) [49, 50, 79] based on WSMO, BPMN [98], and WS-BPEL.

2.3. Conclusion

We have described the technologies and issues of service component integration, explaining that behavioral constraints lead to interactions that cannot be executed. These interactions should be identified and adapted to allow services component interact properly. Service adaptation is emerging as a flourishing area which is attracting attention from numerous researchers [20, 54, 72, 100, 96]. In the next section, we show a motivating example for service adaptation, describing main issues in developing an adaptor for two incompatible service components.

3. The protocol adaptor setting

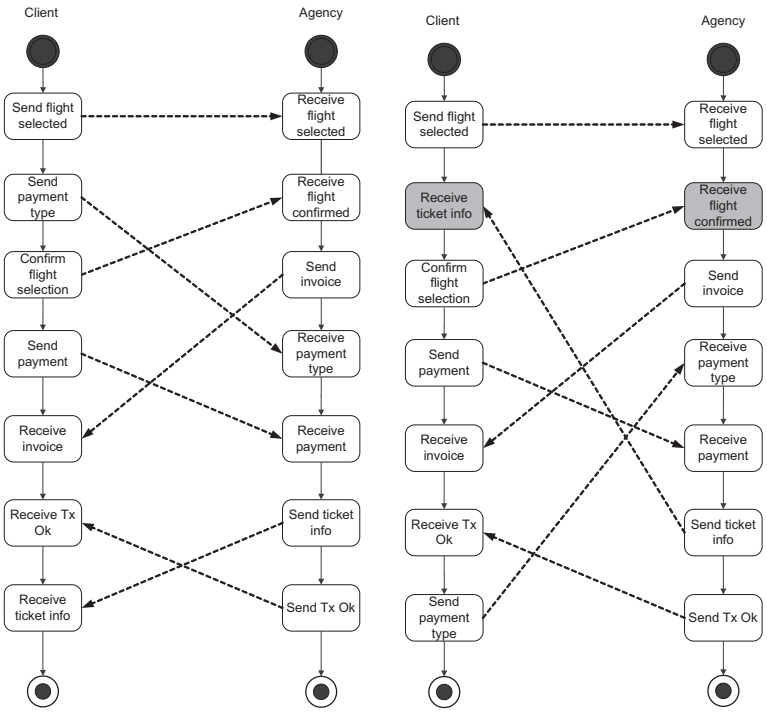
In this section, we give a motivating example for generating protocol adaptors. That example shows two incompatible protocols requiring an adaptor for resolving their behavioral differences. Afterwards, we describe communication semantics and compatibility checking as the main elements for generating protocol adaptors.

3.1. A motivating example

Nowadays, Service Oriented Architectures (SOAs) should be built adopting a federated approach. That is, allowing interoperability of services on-the-fly through dynamic service adaptation [25]. However, it is not easy to generate an adaptor. Firstly, we have to identify protocol mismatches in the composite services.

Figure 1 shows two scenarios with each business protocol by means of a variant of statecharts. The transitions are depicted with solid arrows between each state into the protocols. The interactions are depicted with dotted arrows between the states of the protocols, representing the message exchanges; these dotted arrows are not part of the original statechart [37, 38, 48] notation. In the scenarios, a *client service* invokes operations to reserve flight tickets from an *agency service*. In both scenarios, the protocols mismatch.

Two protocols can properly interact if one service can invoke all operations of the other side and vice versa. The services have no *unspecified receptions* if for each state they reach, if any of them sends a message, the other party will receive it in a future state. However, both protocols



(a) Incompatible protocols (b) Unresolvable deadlock

Figure 1: *Incompatible service protocols to be adapted*

could get stuck while they are invoking operations from one another due to ordering constrains. This is depicted in Figure 1(a) in the second interaction where the *client service* is sending a message that is not accepted for the *agency service*. However, this *deadlock* can be solved if an adaptor receives and stores such a message (`payment type`). Afterwards, the adaptor should send the message when the agency service can accept it.

In order to reconcile deadlocks, a protocol adaptor should be generated for resolving all mismatches in the message exchanges (dotted arrows) and ensuring both services have reached their final state (bull's eye). Five different adaptors generated for the business protocols of the Figure 1(a) are depicted in Figure 2. However, in the next section we show there are more adaptors for this example. In contrast, Figure 1(b) show a deadlock between two business protocols that cannot be resolved. Both services reach a state where they are waiting for a message that the other party is not willing to send. Also, these messages were not sent before. An adaptor cannot generate such messages by itself. This deadlock is depicted with shaded states in Figure 1(b).

Therefore, two services can reach a deadlock when:

- they are sending a message at the same time;
- they are sending a message that the other is not accepting at that time; or
- they are waiting for a message at the same time, and such messages were not sent by them before.

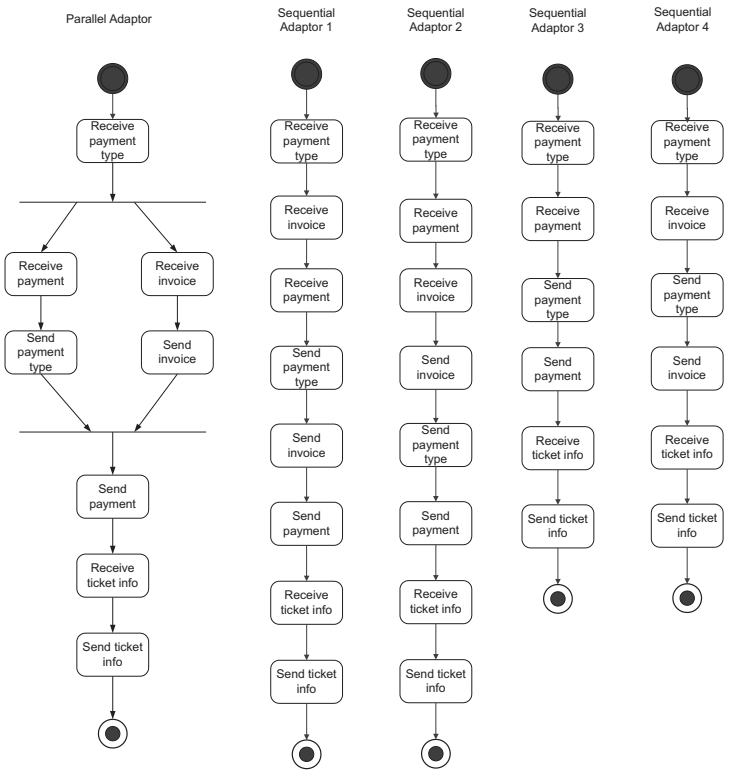


Figure 2: *Sequential and Parallel adaptors generated from a same case*

Two protocols are *compatible* if only if they have no unspecified receptions and are deadlock free [100]. Otherwise, they are incompatible, so an adaptor can be generated to reconcile such differences working as a service in-the-middle. Moreover, that adaptor must be compatible with the incompatible protocols.

Communication semantics [67] and compatibility are two important elements for the generation of protocol adaptors. In the following subsection, we describe the communication semantics used for analyzing the protocols of two interacting composite services. Afterwards, we show the most relevant research in compatibility checking.

3.2. Synchronous and asynchronous semantics

In an asynchronous communication, one service can send a message while the other is unavailable. This communication assumes there is no need for a synchronized connection between the services at all, a service can continue its processing as soon as its request or its response has been sent [57]. Therefore, this communication assumes a queue for storing these messages. If a FIFO queue has a fixed size, it can become full. In that case, an incoming message could either overwrite the oldest message or block the execution of the sender [88].

A synchronous interaction (or blocking) requires that the parties wait until the interaction is concluded; otherwise, the interaction is asynchronous (non-blocking) [3]. A synchronous system

needs well-defined bounds for the time necessary to transmit messages through the communication channel. There is no queue. Moreover, in a synchronous semantics two service protocols must advance atomically from one state to another [69, 101]. With a synchronous semantics it is easier to reason about protocols and define compatibility [100]. However, we can assume that two service protocols are compatible under the synchronous semantics without requiring atomicity in the transitions. That is, the synchronous system require the protocols agree on the total order of messages sent and received between them (execution traces) [100], which is less restrictive than atomicity. Moreover, a run-time arbitrator can be defined to synchronize protocols. It forces the protocols to agree who is to send and who is to receive when they are sending a message at the same time [100].

A technique called *synchronizability analysis* [40] has been defined using Finite State Machines (FSMs) [85]. If a service composition is identified to be synchronizable, it produces the same set of conversations under both asynchronous and synchronous communication semantics. So, asynchronous communication could be replaced with the synchronous communication.

Therefore, although loose coupling between services in a SOA implies an asynchronous semantics [2, 77], and that semantics can be easily implemented, it is difficult to reason about [100]. The synchronous communication semantics is easier to analyze than the asynchronous one. The analysis of some properties such as deadlock are even undecidable using an asynchronous semantics [18].

3.3. Compatibility checking of service protocols

We previously have shown a definition of compatibility. However, many research has been done to check compatibility between two interacting protocols. If two service protocols are incompatible, an adaptor should be defined. Protocol adaptors must synchronize the protocols interactions of two services, following the underlying business logic in a meaningful way [54, 60].

For checking compatibility, mismatch patterns have been defined to capture differences between two business protocols [10]. However, it is not clear whether these mismatch patterns are complete, and the patterns are only informally presented. Bordeaux et al. [16] check the compatibility of two services based on the *substitutability* problem using the Labeled Transition Systems (LTS) notation. Also in [13, 14] substitutability and compatibility checking are focused exclusively on the web-method invocation behavior of services. That assumes a system distributed and asynchronous with multi-threaded components. Moreover, this research provides the constructs of recursion, sequential and parallel composition.

A formal transformation exists of a WS-BPEL description by means of annotated Deterministic Finite State Automata (aDFA) for modeling matchmaking [99]. That transformation represents messages sent by a service at a particular state and the messages supported by the corresponding receiving service. However, a drawback of this approach is the performance since it requires a index structure supporting specific queries in dynamic service discovery scenarios. Another study is the C-composition algorithm [11] based on FSMs where compatibility, equivalence, and replaceability definitions are presented. The performance of that algorithm is polynomial. Dumas et al. [32] presents an algebra of behavioral interfaces allowing the definition of mapping constraints to detect deadlocks. Moreover, a tool based on FSMs is studied for service interface adaptation. Also, algorithms from the game theory have been defined to check compatibility in the interfaces of two components [1, 30, 78], however, they lead to a state explosion problem.

An analysis of usability, compatibility, equivalence and replaceability of WS-BPEL processes is defined based on Petri nets [73, 80] in [62, 64]. These studies allow to define and compose and adaptor for incompatible protocols. Furthermore, in [71] a method also based on Petri nets is studied to automatically generate a compatible partner WS-BPEL process for an input WS-BPEL process. A hybrid approach is used to avoid the state explosion problem [61]. That combines a structural and a behavioral approach. Also, the compatibility, exchangeability and the generation of an adaptor among two components are defined into the SCA context in [63]. This approach is based on Petri nets and a drawback is the performance of computationally expensive analysis steps. Matching behavioral aspects of different processes via state spaces has a state explosion problem due to parallelism. However, there exists a structural and efficient (linear time) heuristic-based for matching WS-BPEL processes defined in [35].

Summarizing, compatibility checking has received a lot of attention [11, 16, 30, 59, 62, 71] for stateful services or components. Some of these works [30, 11] consider unstructured sequential protocols. Other researchers [16, 59, 62, 63, 71] study parallel business protocols which can contain non-determinism. These approaches either use Petri net-based techniques [59, 61, 62, 63, 71] or process algebraic techniques [16] for verifying compatibility. To check compatibility, typically a graph-based representation of the behavior of the business protocols is built, which is expensive to construct for protocols containing parallelism (worst-case time complexity is exponential in the size of the protocol).

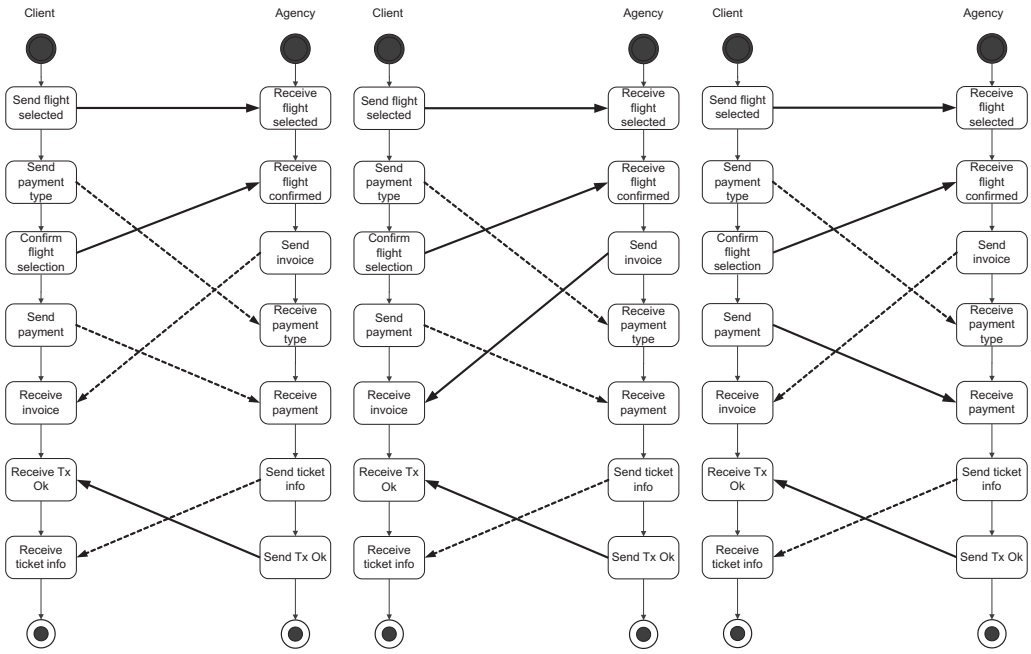
3.4. Conclusion

We have described a motivating example for service adaptation showing two incompatible protocols. We defined compatibility between two service protocols. Also, we have showed that synchronous semantics is easier to reason than the asynchronous one, preserving the compatibility definition. Moreover, for the context of this paper, we described the most important research on compatibility checking highlighting their differences and issues. In the next section, we use the compatibility definition and the synchronous semantics to analyze two service protocols. If these protocols are incompatible, we manually generate an adaptor. We show the main assumptions and decision point by generating that adaptor. Afterwards, we describe the most relevant research in generating semi-automated adaptors, showing the main differences and issues of these approaches. The approaches automate one or more steps in the manual generation of adaptors.

4. Generating protocol adaptors

Protocol adaptors are proposed to reconcile differences on the protocol behavior between two service components plugged together. If two service components need to interact, their business protocols must be compatible [11, 16, 54, 59, 62, 71, 72]. That is, the components should not deadlock while exchanging messages with each other. However, if two service components are incompatible, the question arises how behavioral mismatches can be resolved. One possibility is to change one of the protocols which typically describe built-in behavior of some existing component. It may be a legacy system. So, such a component is not easy to change. Moreover, it is unlikely that the owning party is willing to change its configuration for just one composite service invocation. A more useful solution is to use an adapting protocol between incompatible business protocols. This *adaptor business protocol* can be used to reconcile behavioral mismatches by compensating the differences between the incompatible business protocols.

In the next subsection, we show how adaptors can be generated manually. Also, we illustrate different decision points in developing an adaptor according to a synchronous communication semantics.



(a) Interaction 1 (b) Interaction 2 (c) Interaction 3

Figure 3: *Interaction with a protocol adaptor involved. Solid arrows means no adaptation is needed for that interaction.*

4.1. Manual generation of protocol adaptors

We select the incompatible business protocols depicted in Figure 1(a) to manually generate an adaptor. Three different interaction scenarios are depicted in Figure 3. We model their interactions under the synchronous semantics [11, 72, 100] since it is easier to analyze than the asynchronous case. The analysis in the asynchronous semantics can be undecidable. Also, we assume that there are no mismatches on message types in the protocols. The *client service* can only send a message *m* if it is in a state that enables it to send *m*, and if the *agency service* is in a state that enables it to receive *m*, such that the sending and receipt of a message are considered an atomic action and no queues are required.

In the first interaction of the example (see Figure 3), the client service sends a (`flight selected`) message since the agency service can accept it. Once the message is sent and received, both services advance to the next state atomically. The next interaction cannot be executed. The client service is willing to send the message `payment type` that is not accepted by the agency service since it waits for a `flight confirmation` message. This deadlock is resolved by means of service adaptation. A protocol adaptor receives such a message and stores it temporarily in its memory, forwarding it when the agency service can accept it (see Figure 2).

Once the adaptor receives the message `payment type`, the client service moves to the next state, in which the message `flight confirmation` is sent. The agency service expects a `flight confirmation` message, so no interaction with the adaptor is needed for this interaction. After this interaction both services advance to the next state, reaching a new deadlock. The client service is willing to send a message at the same time as the agency service (see Figure 3). To resolve this deadlock, different approaches might be taken, depending on certain assumptions: communication schema, coordination, precedence, and transitions involved in the three services. The adaptors generated from these assumptions are depicted in Figure 2 and the assumptions are listed in Table 1.

Adaptor name	Synchronous semantics	Arbitrator	Protocol with precedence to send a message
Parallel	Yes	No	Both
Sequential 1 and 2	Yes	No	None
Sequential 3	Yes	Yes	Client service
Sequential 4	Yes	Yes	Agency service

Table 1: *Assumptions for generating protocol adaptors depicted in* Figure 2

Table 1 shows that all components use synchronous communication. If two services are sending a message at the same time, the parallel adaptor assumes that both services send a message at the same time and no arbitrator is used. The sequential adaptor 1 and 2 are also generated with no arbitrator. They send a message to any of the services with no precedence. They are two sequential cases of the parallel adaptor. So, to generate the sequential adaptor 1 we assume that it first receives the message `invoice` from the agency service. In contrast, for the sequential adaptor 2 we assume that it first receives the message `payment` from the client service. In the same way, to build the sequential adaptor 3 and 4, we assume an arbitrator that gives precedence to either the client or the agency service to send a message.

In Figure 3 a solid arrow means adaptation is not needed for such an interaction. The services can exchange messages directly. In this case, three service interactions are always direct, they are not adapted: `Send flight selected`, `Confirm flight selected` and `Transaction Ok`. We get the first interaction when both services do not get stuck. We get the other two direct interactions when the two services get stuck. They do not get stuck with the adaptor. The pair of states where the protocols get stuck are: `send payment type` and `receive flight confirmation`, and `receive Tx Ok` and `send ticket info`. In these states, we choose to adapt one of the interactions following the assumptions. So, in a given pair of states, an interaction is direct if a receiving statement of a protocol matches with a sending statement in the next state of the other protocol.

All adaptors depicted in Figure 2 have the same first state, as we previously detailed it. However, they are differently built when the protocols are in the pair of states: `send payment` and `send invoice`. There, both protocols are sending a message at the same time. We built five different adaptors using the assumptions of Table 1 as follows:

1. The interaction of the business protocols with the parallel adaptor is depicted in Figure 3(a). The parallel adaptor receives the messages `payment` and `invoice`. After that, the ser-

vices move to the next states (`receive invoice` and `receive payment type`) reaching a deadlock. The services are willing to receive a message that no one can send. However, the adaptor stored the messages (`payment type` and `invoice`) previously. It synchronously sends these messages and both services move to the next states reaching a new deadlock. The adaptor sends the message `payment` to the agency service, which moves to the state to send the message `ticket info`. The services again reach a new deadlock. The adaptor must receive that message from the agency service. So, the agency service moves to the next state getting a direct interaction (solid arrow) with the client service. Once they exchange the `transaction Ok` message, both services advance to the next state. The agency service reaches its final state. The client service waits for the adaptor to send the message `ticket info`. After that interaction, both services move to their final state (see Figure 2, parallel adaptor).

2. The business protocols interact with the sequential adaptor 1 (see Figure 2, sequential adaptor 1) as it is depicted in Figure 3(a). This adaptor is a particular case of the parallel adaptor. We assume the adaptor receives the message `invoice` from the agency service that moves to the next state reaching a deadlock. Next, the adaptor receives the message `payment` from the client service that also moves to the next state. Afterwards, both services reach a new deadlock. The adaptor sends the message `payment type` to the agency, which advances to the next state. Then, the adaptor sends the message `invoice` to the client. The remaining interactions were described in the previous item.
3. The business protocols interact with the sequential adaptor 2 (see Figure 2, sequential adaptor 2) as it is depicted in Figure 3(a). This adaptor is a variation of sequential adaptor 1 and another particular case of the parallel adaptor. In the same pair of states, we assume the adaptor receives the message `payment` from the client service that moves to the next state. Here, both services can interact directly. However, we assume this interaction is also adapted since this is also adapted by the parallel adaptor. So, the adaptor receives the message `invoice` from the agency service that also moves to the next state. Afterwards, both services reach a new deadlock. The adaptor sends the message `payment type` to the agency, which advances to the next state: (`receive invoice` and `receive payment type`) Then, the adaptor sends the message `invoice` to the client. The next interactions correspond to those described in the first item.
4. The business protocols interact with the sequential adaptor 3 (see Figure 2, sequential adaptor 3) as it is shown in Figure 3(b). The arbitrator gives precedence to the client service to send a message (see Table 1). So, the adaptor receives the message `payment` and the client advances to the next state. Here, both services interact directly and adaptation is not necessary. The agency sends the message `invoice` directly to the client and both services move to the next state. Next, they reach a deadlock since both services are willing to receive different messages: `transaction Ok` and `payment type`. The latter is stored in the adaptor. It sends the message `payment type` to the agency that moves to the next state. After that, both services reach a new deadlock. The remaining interactions were explained in the first item.
5. The sequential adaptor 4 (see Figure 2, sequential adaptor 4) interacts with the business protocols as it is depicted in Figure 3(c). The arbitrator gives precedence to the agency service to send a message (see Table 1). Thus, the adaptor receives the message `invoice` and the agency moves to the next state. The services reach a deadlock. We assume that the adaptor sends the message `payment type` to the agency service that advances to the next state. Here, adaptation is not needed and both services interact directly exchanging the

message `payment`. After that, both protocols move to the next state. The adaptor sends the message `invoice` to the client that advances to the next state. The remaining interactions must be treated as the first item.

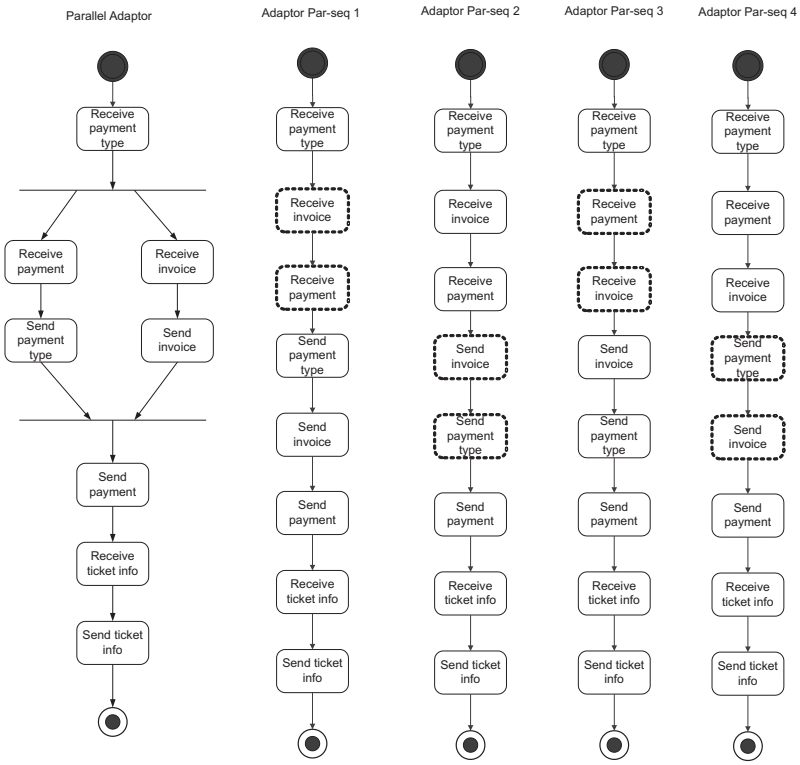


Figure 4: *Four sequential adaptor deduced from the Parallel one*

We showed the diversity of actions taken for adapting an interaction if both protocols are sending a message at the same time: `Send payment` and `Send Invoice`. Also, we showed how difficult it is to reconcile behavioral mismatches of two business protocols (see Table 1). The previous example shows five different protocol adaptors generated from the same business protocols. Sequential adaptors 3 and 4 have less states than sequential adaptors 1 and 2, and the parallel adaptor. They exchange less messages, so both might have better performance. Moreover, the five adaptors differ on what they assume about the interaction if both services are sending a message at the same time. Sequential adaptors receive a proper message from either the client or the agency service, whereas the parallel adaptor receives both messages (see Table 1).

Sequential adaptors 1 and 2 are special cases of the parallel adaptor. Also, there are other two special cases. We show four sequential cases of the parallel adaptor in the Figure 4. We show differences among them in the states with dotted borders. Moreover, other combinations may be possible (for instance `receive payment, send payment type, receive invoice, send invoice`) but these are not allowed by the environment. Sequential adaptors 1 and 2 in the Figure 2 corresponds to the par-seq 1 and par-seq 3 of the Figure 4. In this hypothetical example we

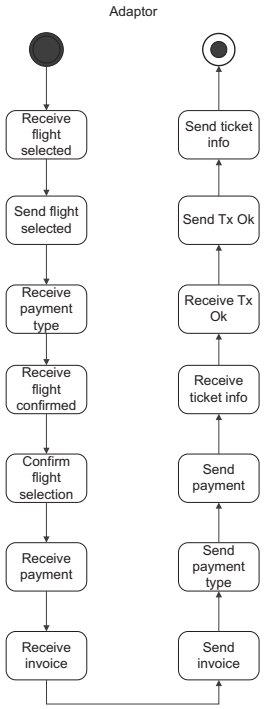


Figure 5: Sequential Protocol Adaptor generated for all interactions of Figure 1(a)

built manually seven different protocol adaptors from the same interacting business protocols as it was shown in Figures 2 and 4.

However, there exists another adaptor deduced by adapting all interactions of the Figure 3(a). That adaptor is depicted in Figure 5. That adaptor contains the same states of the Sequential adaptor 2 (see Figure 2), and the states without needing adaptation: `Send flight selected`, `Confirm flight selected` and `Transaction Ok`. So, we could build other variants from that adaptor by adding the states not needing adaptation to the other five sequential adaptors.

Therefore, we could get other six sequential adaptors from the previously deduced. In that case, we could build twelve different sequential adaptor and a parallel one. However, we are interested in building an adaptor only for those interactions needing adaptation. So, we wish to pick the best adaptor that fits to given case. One heuristic to choose that adaptor should be the minimal number of states, assuming that implies a better performance.

A manual generation of adaptors becomes tedious and error prone as the number of interactions is increasing, even more when the service protocols have parallel behavior rather than only sequential. In the next subsection, we examine the most important research on semi-automated generation of adaptors for service components.

4.2. Semi-automated generation of protocol adaptors

First, we give a general overview of the existing approaches for generating adaptors semi-automatically. Next, we describe the most important semi-automated approaches, focusing on

their similarities and differences. Finally, we discuss a summarizing table of these approaches, showing their main differences.

4.2.1. A general overview of the existing approaches

Many service-oriented interactions are fully automated and require no or little user interaction. Since modeling is time consuming and error-prone [71], specifying adaptors manually considerably slows down the development. Also, this hampers the performance of the business processes of the involved organizations. A more fruitful approach is to generate these adaptors automatically. Approaches that focus on automatic generation of adaptors for components are [17, 19, 20, 54, 72, 78, 96, 100] but they are not fully automated in the sense that they need additional input from a user. So, they are semi-automated rather than automated approaches.

Yellin and Strom [100] augment software interfaces with sequential protocols, represented as Finite State Machines (FSMs). Also, they define the notion of a software adaptor that can act as a bridge between two incompatible protocols. An adaptor is similar to a protocol, but it has in addition a restricted memory to store and retrieve parameters of received messages. Moreover, they define a high-level mapping language that can be used to relate parameters of messages sent or received by the two components. From a high-level declarative adaptor specification, automatically an adaptor for components with incompatible protocols can be synthesized.

The approach by Brogi et al. [17, 19] extends the Yellin and Strom [100] approach, but it does not consider adaptors with memory, unlike Yellin and Strom [100]. They use CCS-like process algebra [68] to specify protocols rather than FSMs. The high-level mapping language they use is more simple than the one of Yellin and Strom [100]. Next, they do not require the use of arbitrators. Though their adaptors can contain parallelism, the algorithm for deriving adaptors as presented in [17] is exponential, and thus not tractable.

Motahari et al. [72] also extends the Yellin and Strom [100] approach to generate a sequential adaptor automatically. This approach gives suggestions to a developer for resolving deadlocks using a mismatch tree. That tree is based on data traces and it shows decision points to the developer in each deadlock reached by the service components. If there is not a feasible path, the deadlock is not resolved.

Other recent work also based on FSMs is given by Wang et al. [96]. They define a run-time service adaptation method for describing a set of mapping rules containing different interaction scenarios. So, that method resolves the interface and behavioral mismatches among two interacting services.

Brogi et al. [20] define a method to automatically generate WS-BPEL adaptors. That approach transforms the protocols of two interacting services in YAWL workflows [91]. The adaptor is built from such workflows by building Service Execution Trees (SETs), then it is transformed into a WS-BPEL specification. In contrast, Kumar and Shan [54] define analytical algorithms to generate WS-BPEL adaptors automatically based on compatibility patterns.

Passerone et al. [78] extend the work of Alfaro and Henzinger [30] by presenting an approach for synthesizing memoryless adaptors based on classical game theory. The complexity is linear in the size of the game structure (state space). However, deriving the game structure from a protocol containing parallelism is exponential in the size of the protocol (state explosion problem). Also, the constructed adaptors are memoryless.

Other related work [10, 32, 84] focuses on semi-automated specification of adaptors. Benatallah et al. [10] identify several mismatch patterns for incompatible business protocols and specify for each pattern a corresponding piece of adaptor code that can resolve the mismatch. However, it is not clear whether these mismatch patterns are complete, and the patterns are only

informally presented. Dumas et al. [32] study the problem of specifying adaptors declaratively. They present a tool for executing adaptor specifications. Schmidt and Reusner [84] focus on somewhat different problem: they define adaptors for the case that a sequential protocol communicates with more than one other sequential protocol. Such a bridge adaptor converts a set of protocols into a single protocol that interfaces with the protocol at the other side.

We next describe the most important research efforts in this area [20, 54, 72, 96, 100] in more detail. We focus on research done in the field of Web services, with the exception of [100] which is from the pre Web services era, but has laid the foundation for most of the other approaches we next discuss.

4.2.2. Heuristic generation of adaptors based on pattern analysis

Kumar and Shan [54] defined a method to generate WS-BPEL adaptors via compatibility pattern verification. This research assumes a synchronous semantics and no syntactic mismatches in the interfaces of the services. However, algorithms are specified using pseudocode rather than FSMs [85], or Process Algebra[8, 7, 68, 81], or Petri nets [73, 80]. Services are structured and they can contain parallelism, loops, choices and sequences. These basic process patterns are deduced from the corresponding names used in WS-BPEL. Moreover, this approach builds a matrix to show the compatibility among the patterns. This differs from the mismatch patterns defined in [10].

A normalization procedure deletes internal activities from the protocols. Also, it merges and simplifies non-loop patterns in the protocols. However, loop patterns are not normalized. Afterwards, the approach verifies if these protocols are compatible. Furthermore, this approach assumes that if two services are compatible they do not need an adaptor for asynchronous communication. The services send a message without waiting for a reply. So, every message is put in a random access buffer until it is requested for a service. In contrast, for synchronous communication, the services send a message and wait for a reply, so they can reach a deadlock. However, with asynchronous communication, a deadlock is also possible. If this deadlock can be resolved, the approach builds an adaptor.

An algorithm checks compatibility between two protocols once they are normalized. It receives two protocols as arguments and it works recursively by decomposing them. First, the algorithm checks that each activity in a protocol has its corresponding dual in the other protocol. So, if an activity has a send statement, its corresponding dual is the same activity with a receive statement; and viceversa. Secondly, the algorithm checks if there exists a deadlock that cannot be resolved. That is, the services are waiting a message at the same time, and such messages were not send before by any one. For checking a deadlock, the algorithm transforms the protocols in two directed graphs. It changes loop structures into sequences, it changes the activities into nodes, and it changes the links into directed vertices. Moreover, the interactions are changed into directed vertices transforming the two graphs in a new one. There exists a deadlock if the algorithm finds a cycle in that directed graph. Therefore, the services are incompatible and there is not an adaptor for them. Thirdly, the algorithm decomposes the choice and loop patterns of the protocols and it checks if the corresponding branches are compatible. Otherwise, the services are incompatible. If the protocols are incompatible, it is not possible to build an adaptor.

The compatibility checking reduces the search space by analyzing the patterns based on the matching matrix and prerequisite rules. If the algorithm verifies that two protocols are compatible, the approach generates an adaptor using the Pattern-based Algorithm (PBA). Alternatively, it generates a minimal adaptor using the PBA-Min algorithm. The PBA algorithm starts with the innermost loop or choice patterns in each process, then it creates an adapter for this structure.

So, the remaining sub-processes have only sequence and parallel patterns. This approach resolves deadlocks using the same heuristic shown in the Table 1 for generating a parallel adaptor. Moreover, the PBA-Min algorithm generates a smaller adaptor. That algorithm implements the heuristic of the Table 1 for generating either the sequential adaptor 3 or 4. So, it chooses the best send activity to adapt using a Minimum Receive Distance (MRD) and a Minimum Send Distance (MSD). However, an arbitrator is not used here. Consequently, an issue in the synchronous semantics appears since one of the protocols get blocked while trying to send a message. This drawback could force the environment to change its synchronous semantics if no arbitrator is used. The generated adaptor uses a random memory to store the messages. If the adaptor is null, the protocols do not need adaptation. If we apply the PBA algorithm to our motivating example (see Figure 3(a)) we get the parallel adaptor depicted in Figure 2. The minimal adaptor corresponds to the Sequential adaptor 3 of the same figure.

4.2.3. Generation of adaptors based on interface mappings

Yellin and Strom [100] is one of the most relevant works in building semi-automated adaptors by defining compatibility by means of a high-level declarative specification. This approach uses augmented interface descriptions containing message (method) signatures for both messages sent and messages received. Also, it defines protocols with the legal sequences of messages exchanged among two services. Moreover, it is based on a synchronous semantics where sequential protocols are represented as finite state machines (FSMs). There are only one send or receive statement per state. An arbitrator in the synchronous semantics is used to force both services agree who is to send and who is to receive, when they are sending a message at the same time (see Table 1).

The approach supports the following steps:

- the specification of abstract protocols of interaction for a set of components;
- the definition of how to realize these protocols; and
- the implementation of the synchronous semantics with an arbitrator.

If the protocols of two components are incompatible, the approach can generate an adaptor compatible with both protocols, even if the protocols do not support the same set of messages.

The adaptor is modeled as a FSM with:

- a finite set of states;
- a finite set of typed memory cells where each parameter received is stored in exactly one memory cell; and
- transition rules consisting of state transitions involving either a sending or a receiving message, with memory actions to store and synthesize such a message.

The synchronous semantics is implemented without requiring the two components send and receive messages atomically. Instead, the approach requires the two components always agree on the order of sending and receiving the messages (or execution traces). This assumption simplifies the reasoning about protocols. A single arbitrator is used, rather than one for each pair of interfaces. So, it needs to potentially be aware of all messages exchanged between the three parties. There is not a separate arbitrator for the collaborations between the adaptor and each

component. An adaptor stores only one parameter of a message for each transition in the protocol. All messages exchanged by the components go through the adaptor. Its behavior is governed by its transition rules. However, in a more general case, two protocols could send an unbounded number of messages to the adaptor. If the adaptor stores an unbounded number of messages, this would lead to undecidable results.

There exists an algorithm for checking compatibility between the adaptor and the protocols. Two protocols are compatible if only if they have no unspecified receptions and are deadlock free. Otherwise, they are incompatible, so an adaptor must be generated for resolving such differences. Furthermore, the approach checks the compatibility between the adaptor initially generated and the protocols. Thus, an adaptor is compatible with the protocols if only if they have no unspecified receptions and are deadlock free.

The approach defines an Interface Mapping as a very concise declarative specification relating the parameters and messages exchanged by the protocols. Moreover, the approach either generates a well-formed adaptor valid with that Interface Mapping or determines that no such adaptor exists. That algorithm works in two phases. In the first phase, it constructs an initial adaptor marking all states and possible transitions with the protocols. In the second phase, it removes deadlocks and unspecified receptions from that initial adaptor. If the final adaptor is empty, there is not a valid adaptor for the given Interface Mapping. Otherwise, there exists a valid adaptor for that Interface Mapping. The approach generates a sequential adaptor for all interactions among two services. The adaptor obtained by applying this approach to the business protocols of the Figure 3(a) is depicted in Figure 5. That adaptor is manually constructed in the previous section.

4.2.4. Generation of adaptors using mismatch trees for resolving deadlocks

Motahari et al [72] have extended the work of Yellin and Strom [100] to generate a sequential adaptor resolving deadlocks with the input of a developer.

The approach uses the algorithm in [100] to generate the adaptors. So, a FSM formalization using the synchronous semantics is the basis. Firstly, this study analyzes interface level mismatches between messages in the interfaces of two protocols. Unlike [17, 20, 100], this assumes the Interface Mappings are not provided, but it helps the developer in providing one. The approach assumes the matching is not properly realized without considering the ordering constraints. Secondly, the approach analyzes if there are mismatches at the protocol level: unspecified receptions and deadlock. However, the approach assumes that unspecified receptions can be resolved using [17, 20, 100]. Instead, that focuses on resolving deadlocks.

The approach has no explicit algorithm for compatibility checking. It focuses on providing support to identify interface-level mismatches. That matching is based on schema matching using three heuristics: pair-wise matching of schema messages, adding message names into the schema, and considering the message type (input/output) into the schema. Also, the approach provides support for the identification of protocol-level mismatches to generate an adaptor. It uses the compatibility definition of [100] identifying unspecified receptions and deadlocks.

The approach uses [100] to generate the adaptor. Then, the adaptor has a finite set of typed memory cells, it stores one copy of any parameter (one memory cell). The approach handles deadlocks by Progressive User Interaction and by generating a Mismatch Tree for all mismatches leading to deadlock. The first method prompts the developer with the messages responsible of each deadlock found by the algorithm for generating an adaptor. Also, this provides feasible information to resolve such a deadlock, so the developer provides the mappings to resolve the

deadlock. Otherwise, the developer remove this deadlock from the adaptor. This method proceeds until the algorithm finds another deadlock. The second method generates a Mismatch Tree using a what-if analysis for each deadlock found by the algorithm that is generating the adaptor. A Mismatch Tree represents all deadlocks and the messages involved in each deadlock. So, the developer can make better decision than with the other method. The developer can resolve a deadlock constructing the engaged message in a deadlock based on evidences (interface-based inference and execution logs). Otherwise, the deadlock is tagged as non-resolvable. So, in the second phase of the algorithm [100] such deadlocks are removed from the adaptor.

The drawbacks of the first method are: too many interactions with the developer, and that the developer may not make a good decision. In contrast, Mismatch Trees have the advantage that they represent all possible deadlocks. These allow the developer to make informed decisions: update the interface mappings to resolve some deadlocks, or tag some of the deadlocks as non-resolvable. However, although [54, 100] assume there are no mismatches at the interface level, this approach is not more expressive. The deadlock of Figure 3(b) is tagged as non-resolvable by applying this approach, so no adaptor can be generated. Moreover, this approach generates the same adaptor as [100] and also the one manually generated before. That is depicted in the Figure 5 for the business protocols of the Figure 3(a).

4.2.5. Generation of adaptors based on service execution trees

The approach of Brogi et al. [20] defines a method to generate semi-automatically WS-BPEL adaptors. This method uses service contracting, WSDL signatures and YAWL [91] as intermediate language to provide a partial description of service behaviors.
It has four phases:

- *Service translation* phase, it translates the WS-BPEL descriptions of two services into their corresponding YAWL workflows.
- *Adaptor generation* phase, it generates a Service Execution Tree (SET) for the adaptor from the merging of corresponding SETs of the service protocols.
- *Lock analysis* phase, it verifies whether the adaptor (YAWL workflow-based) generated deadlocks with the services, if it does, the adaptation has failed; otherwise it is successful or partial.
- *Adaptor deployment* phase, it deploys the YAWL workflow of the adaptor as a WS-BPEL process.

This approach assumes synchronous and asynchronous communication semantics for interacting services. Also, parallel, loops and other WS-BPEL structures are present. There is not a formal definition of compatibility of two protocols in this approach unlike [100]. Once a SET of an adaptor is generated, if there is not a successful trace, the adaptation fails. Otherwise, this SET is transformed into a YAWL workflow. After that, the compatibility is checked by searching deadlock-free traces between the services and the adaptor. If there are some or no deadlocks, the adaptation succeed completely or partly, otherwise it is failed.

A SET of a WS-BPEL process is generated, also considering loops in the process, by a *reachability analysis* of its corresponding YAWL workflow obtained during the service translation phase. Moreover, a SET of a service contains all messages exchanged of this service with the other one. A dual SET is founded by defining a SET of a service. Afterwards, the dual SET

of two services are merged to generate a SET of the adaptor. This SET matches the synchronous and asynchronous communications of the activities and the data-flow dependencies. Such dependencies define on which order the messages are stored and forwarded by the adaptor. Also, these are either constrained or unconstrained. Moreover, there is no memory restrictions for the adaptor. Therefore, if such a SET has at least one successful trace, then a YAWL workflow is built; otherwise, no adaptor exists. After that, the deadlocks are checked between the adaptor and the services. If there are no traces deadlock-free, the adaptation is failed; otherwise the adaptation is successful. The adaptation is partial if there are interactions cannot be adapted. Finally, by applying this approach to the business protocols of Figure 3(a) it generates the same adaptor as [72, 100] and also the one manually generated before. That is depicted in the Figure 5.

4.2.6. Generation of adaptors based on a run-time method

Wang et al. [96] have defined a run-time service adaptation method. This approach is based on Finite State Machines (FSMs) for developing a run-time adaptor between two incompatible services. That approach selects and chains mapping rules to reconcile the mismatches. Such mapping rules describe a transformation between one or multiple source message types and a target message type, then they are chained to produce the message expected by the services based on previously intercepted messages. A mapping rules repository is used for reconciling the differences of both interacting services. That is a collection of rules generated from a number of adaptation scenarios. The adaptor is defined by the FSMs of the two services and a mapping rules repository. In the adaptation cycle, an adaptor intercepts all messages exchanged among both services. Then, it triggers an internal cycle to forward the corresponding messages without interference from the external side. When the adaptor intercepts and stores a message, the state of the sending service is updated. If the message is expected by the other service, it is forwarded; otherwise, the message type is checked and a firing sequence is looked for. Afterwards, the adaptor executes such a sequence and some messages are sent to the other side. This procedure is repeated until no more firing rules can be found.

Unlike [100], the approach has no definition of compatibility. Compatibility is not checked between two protocols before to generate the adaptor, instead deadlock and information loss are detected at run-time. A deadlock scenario occurs if the adaptor is waiting indefinitely for more messages to arrive. This is detected if the adaptor requires to forward any message when it terminates. An information loss scenario occurs if a message intercepted and stored by the adaptor is not released at the end of the adaptation cycle.

This approach provides a run-time adaptation method to determine if differences of a pair of services can be resolved. However, sufficiency of the mapping rules set has not yet been determined. Also, the definition of the data transformation function for building mapping rules is not provided. Moreover, compatibility at the interface level is not checked. This method differs from those previously explained since it is based on run-time chaining and firing of mapping rules. In contrast, other approaches are based on static compatibility analysis and/or adaptor synthesis. Moreover, as previous approaches, by applying this approach to the business protocols of Figure 3(a) it generates the same adaptor as [20, 72, 100] as depicted in the Figure 5, but *at run-time*.

4.2.7. Discussion

Table 2 summarizes the approaches previously described. All approaches assume synchronous semantics. However, Brogi et al. [20] and Kumar and Shan [54] are also defined for asynchronous communication.

Characteristics		Kumar and Shan [54]	Yellin and Strom [100]	Motahari et al. [72]	Brogi et.al. [20]	Wang et al. [96]
Comm. semantics	Synchronous	✓	✓	✓	✓	✓
	Asynchronous	✓			✓	
Execution Model	FSM		✓	✓		✓
	Petri Nets				✓	
	Pseudocode	✓				
Process type	Structured	✓				
	Non-structured		✓	✓	✓	✓
Process choices	Deterministic	✓	✓	✓	✓	✓
	Non-deterministic					
Process concurrency	Sequential	✓	✓	✓	✓	✓
	Parallel	✓			✓	
Time of computation	Design-time	✓	✓	✓	✓	
	Run-time					✓
Main approach element	Interface Mapping		✓		✓	
	Pattern Analysis	✓				
	Mapping Rules		✓	✓		✓
	Service Execution Trees				✓	
	Mismatch Trees			✓		
	Message transformation					✓
Compatibility	Input protocols	✓	✓	✓		
	Prot. and adaptor	✓	✓	✓	✓	✓
Adaptor memory	MR repository					✓
	Typed cell-memory set		✓	✓		
	Random memory-buffer	✓				
Deployment	Arbitrator		✓	✓		
	Synchronous		✓	✓		
	Asynchronous	✓	✓	✓	✓	✓

Table 2: *Comparison of current approaches for semi-automated protocol adaptor generation*

Kumar and Shan [54] define an heuristic approach, while the other are either based on FSMs or Petri net theory. Moreover, Kumar and Shan [54] and Brogi et al. [20] have defined algorithms to generate WS-BPEL adaptors for parallel protocols. The other approaches are only defined for sequential protocols.

Wang et al. [96] have defined a behavioral and syntactic run-time adaptation method, however, other approaches define a design-time behavioral adaptation method. While Kumar and Shan [54] assume structured processes, the other approaches assume unstructured processes. Moreover, all approaches assume deterministic choices.

The approaches have different elements in the execution model. Unlike [20, 54, 100, 96], Motahari et al. [72] assumes the interface mappings are not provided, so that approach helps the developer in providing one. FSMs-based approaches use mapping rules for an adaptor synthesizes messages, but only Wang et al. [96] uses message transformation at run-time for building an adaptor. However, Brogi et al. [20] generates an adaptor using Service Execution Trees. While Kumar and Shan [54] checks compatibility of two protocols using a Pattern matrix, Motahari et al. [72] identifies deadlocks using Mismatch Trees.

On the other hand, all approaches construct an adaptor compatible with the interacting protocols. However, Yellin and Strom [100] is the only approach that formally defines compatibility between the adaptor and the protocols. Also, that approach removes all deadlocks and unspecified receptions from an adaptor. Brogi et al. [20] checks if an adaptor is deadlock-free, the adaptation is successful; otherwise the adaptation is partial or it is not successful. Moreover, an adaptor generated by the Kumar and Shan [54] is deadlock-free by construction.

Brogi et al. [20] do not mention where the adaptor stores the messages exchanged by the protocols. In contrast, Yellin and Strom [100] generate an adaptor using a finite set of typed memory cells, and Kumar and Shan [54] assume an adaptor uses a random memory buffer. Wang et al. [96] assume a Mapping Rule repository.

Yellin and Strom [100] and Motahari et al. [72] implement an arbitrator in the execution model for the synchronous communication. Other approaches are implemented using asynchronous communication.

Indeed, all approaches have valuable contributions aiming for a fully-automated generation of adaptors.

4.3. Conclusion

In this section, we have manually constructed twelve sequential protocols adaptors and a parallel one for the motivating example. Three out of thirteen adaptors can be constructed by the described semi-automated approaches. That set can be made more diverse if we add more states to a given adaptor, building more variants. We wish to choose the best adaptor that fits to a given case. Heuristics to select an adaptor could be the ease of generation, time of computation and deployment, and performance. That performance could be related to either the number of messages exchanged by the service components or the number of states in the adaptor model.

On the other hand, the detailed approaches have similarities and differences that have been highlighted in the summarizing Table 2. However, some blank spots can be found between these approaches, which are discussed in the next section.

5. Conclusions and further work

Service composition enables service interoperability between different companies. However, if two service components are incompatible, an adaptor must be generated to reconcile their

differences. That adaptor must be compatible with these service components; otherwise, that adaptor does not exist. An adaptor could be generated using manual or semi-automated approaches. We have show twelve different sequential adaptors manually generated from a same case. Also, we manually generated a parallel one. Only three of these adaptors can be generated using the existing semi-automated approaches applied to the same case. This suggest that the existing approaches can be extended to improve the diversity and quality of the adaptors they generate. These existing approaches have similarities and differences highlighted in the Table 2. That table details the state of art in the area of semi-automated generation of adaptors for service components.

In this survey, we found that most of the semi-automated approaches focus on sequential protocols rather than parallel. Also, all approaches assume deterministic external choices. However, non-deterministic internal choices influence the external environment. These process choices must be assumed by a new semi- or fully-automated approach. Moreover, only one described approach generates an adaptor at run-time; all other approaches generate an adaptor at design-time. An hybrid approach to generate an adaptor is interesting to explore, combining the main elements of the described approaches. That combination should generate an adaptor at run-time focused only on protocol incompatibilities rather than on interface mismatches. Also, a new approach should consider structured and non-structured processes.

We plan to design the algorithms for overcoming the described gaps, aiming for a fully-automated approach for generating protocol adaptors. Our starting point will be to develop a hybrid semi-automated method for generating a set of protocol adaptors at run-time, providing the adaptor that fits best to a given case.

References

[1] B.T. Adler, L. de Alfaro, L. Dias da Silva, M. Faella, A. Legay, V. Raman, and P. Roy. Ticc: A tool for interface compatibility and composition. In *Computer Aided Verification*, volume 4144/2006 of *Lecture Notes in Computer Science*, pages 59–62. Springer Berlin / Heidelberg, August 2006.

[2] G. Alonso. Challenges and opportunities for formal specifications in service oriented architectures. In *PETRI NETS '08: Proceedings of the 29th international conference on Applications and Theory of Petri Nets*, pages 1–6, Berlin, Heidelberg, 2008. Springer-Verlag.

[3] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer, 2004.

[4] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guzar, N. Kartha, C. Kevin Liu, R. Khalaf, D. Kning, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, and A. Yiu. Web services business process execution language WS-BPEL version 2.0, April 2007. OASIS Standard.

[5] S. Angelov and P. Grefen. An e-contracting reference architecture. *Journal of Systems and Software*, 81(11):1816–1844, 2008.

[6] A. Arsanjani. Introduction: Developing and integrating enterprise components and services. *Communications of the ACM*, 45(10):30–34, 2002.

[7] J.C. M. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2-3):131–146, 2005.

[8] J.C.M. Baeten and W.P. Weijland. *Process algebra*. Cambridge University Press, New York, NY, USA, 1990.

[9] BEA, Cape Clear Software, IBM, Interface21, IONA Technologies, Oracle, Primenton Tech., Progress Software, Red Hat, Rogue Wave, SAP AG, Siemens AG, Software AG, Sun Microsystems, Sybase, TIBCO, and Zend Technologies. Service component architecture (SCA). `http://www.osoa.org`, 2008.

[10] B. Benatallah, F. Casati, D. Grigori, H.R.M. Nezhad, and F. Toumani. Developing adapters for web services integration. In O. Pastor and J. Cunha, editors, *CAiSE*, volume 3520, pages 415–429. Springer, 2005.

[11] B. Benatallah, F. Casati, and F. Toumani. Representing, analysing and managing web service protocols. *Data Knowledge Engineering*, 58(3):327–357, 2006.

[12] B. Benatallah, M. Dumas, and Q.Z. Sheng. Facilitating the rapid development and scalable orchestration of composite web services. *Distributed and Parallel Databases*, 17(1):5–37, 2005.

[13] D. Beyer, A. Chakrabarti, T.A. Henzinger, Dirk Beyer, Arindam Chakrabarti, and Thomas A. Henzinger. Web service interfaces. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 148–159, New York, NY, USA, 2005. ACM.

[14] D. Beyer, A. Chakrabarti, T.A. Henzinger, and S.A. Seshia. An application ofweb-service interfaces. In *IEEE International Conference on Web Services (ICWS 2007)*, pages 831–838, Salt Lake City, Utah, USA, July 2007. IEEE Computer Society.

[15] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and Orchard. D. Web services architecture. W3C, February 2004. `http://www.w3.org/TR/ws-arch/`.

[16] L. Bordeaux, G. Salan, D. Berardi, and M. Mecella. When are two web services compatible? In M. Shan, U. Dayal, and M. Hsu, editors, *Lecture Notes in Computer Science*, pages 15–28. Springer, 2005.

[17] A. Bracciali, A. Brogi, and C. Canal. A formal approach to component adaptation. *Journal of Systems and Software*, 74(1):45–54, 2005.

[18] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[19] A. Brogi, C. Canal, and E. Pimentel. On the semantics of software adaptation. *Science of Computer Programming*, 61(2):136–151, July 2006.

[20] A. Brogi and R. Popescu. Automated generation of BPEL adapters. In A. Dan and W. Lamersdorf, editors, *Lecture Notes in Computer Science*, pages 27–39. Springer, 2006.

[21] J. de Bruijn and H. Lausen. Web service modeling language - WSML. W3C, June 2005. `http://www.w3.org/Submission/WSML`.

[22] T. Bultan and X. Fu. Specification of realizable service conversations using collaboration diagrams. *Service Oriented Computing and Applications*, 2(1):27–39, 2008.

[23] C. Canal, J.M. Murillo, and P. Poizat. Software adaptation. *L'Objet, Hermes - Lavoisier*, 12(1):9–31, 2006. 1st International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'04).

[24] F. Casati. Service-oriented computing. *SIGWEB Newsl.*, 2007(Winter):1, 2007.

[25] D. Chen, G. Doumeingts, and F. Doumeingts. Architectures for enterprise integration and interoperability: Past, present and future. *Computers in Industry*, 59(7):647–659, September 2008.

[26] R. Chinnici, J. Moreau, A. Ryman, and S. Weerawarana. Web services description language WSDL version 2.0. W3C, June 2007. `http://www.w3.org/TR/wsdl20/`.

[27] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language WSDL version 1.1. W3C, 2001. `http://www.w3.org/TR/wsdl`.

[28] L. Clement, A. Hately, and T. Rogers. UDDI version 3.0.2. OASIS, October 2004.

[29] Crosswork. Cross-organisational workflow formation and enactment, IST no. 50759. `http://www.crosswork.info/` Accessed 2008.

[30] L. de Alfaro and T.A. Henzinger. Interface automata. In *ESEC/FSE-9: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 109–120, New York, NY, USA, 2001. ACM.

[31] M. Dumas, B. Benatallah, and H.R. Motahari Nezhad. Web service protocols: Compatibility and adaptation. *Bulleting of the IEEE Computer Society Technical Committee on Data Engineering*, 31(3):40–44, 2008.

[32] M. Dumas, M. Spork, and K. Wang. Adapt or perish: Algebra and visual notation for service interface adaptation. In J.L. Dustdar, S.; Fiadeiro and A.P. Sheth, editors, *Lecture Notes in Computer Science*, pages 65–80. Springer, 2006.

[33] M. Dumas, W.M.P. van der Aalst, and A.H. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software Through Process Technology*. Wiley-Interscience, 2005.

[34] S. Dustdar and W. Schreiner. A survey on web services composition. *Int. J. Web Grid Serv.*, 1(1):1–30, 2005.

[35] R. Eshuis and P. Grefen. Structural matching of BPEL processes. In *Web Services, 2007.ECOWS '07.Fifth European Conference on*, pages 171–180, November 2007.

[36] R. Eshuis, P. Grefen, and S. Till. Structured service composition. In *Business Process Management*, pages 97–112, 2006.

[37] R. Eshuis, D.N. Jansen, and R. Wieringa. Requirements-level semantics and model checking of object-oriented statecharts. *Requir. Eng.*, 7(4):243–263, 2002.

[38] Rik Eshuis. Reconciling statechart semantics. *Science of Computer Programming*, In Press, 2008.

[39] C. Ferris and J. Farrell. What are web services? *Communications of the ACM*, 46(6):31–, 2003.

[40] X. Fu, T. Bultan, and J. Su. Synchronizability of conversations among web services. *IEEE Transactions on Software Engineering*, 31(12):1042–1055, 2005.

[41] P. Grefen. Workflow management in electronic commerce. In *Conceptual Modeling ER 2002*, Lecture Notes in Computer Science, pages 13–14. Springer Berlin / Heidelberg, January 2002.

[42] P. Grefen. Towards dynamic interorganizational business process management. In *WETICE*, pages 13–20, 2006.

[43] P. Grefen, K. Aberer, Y. Hoffner, and H. Ludwig. CrossFlow: Cross-Organizational Workflow Management in Dynamic Virtual Enterprises. *Computer Systems Science & Engineering*, 1(5):277–290, 2000.

[44] P. Grefen, H. Ludwig, A. Dan, and S. Angelov. An analysis of web services support for dynamic business process outsourcing. *Information and Software Technology*, 48(11):1115–1134, November 2006.

[45] P. Grefen, N. Mehandjiev, G. Kouvas, G. Weichhart, and R. Eshuis. Dynamic business network process management in instant virtual enterprises. BETA Working Paper Series WP198, Eindhoven University of Technology, 2007.

[46] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, H.F Nielsen, A. Karmarkar, and Y. Lafon. SOAP version 1.2 part 1: Messaging framework (second edition). W3C, April 2007. `http://www.w3.org/TR/soap12-part1/`.

[47] A. Halevy, N. Ashish, D. Bitton, M. Carey, D. Draper, J. Pollock, A. Rosenthal, and V. Sikka. Enterprise information integration: successes, challenges and controversies. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 778–787, New York, NY, USA, 2005. ACM.

[48] D. Harel. Statecharts: A visual formulation for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[49] M. Hepp, F. Leymann, J. Domingue, A. Wahler, and D. Fensel. Semantic business process management: A vision towards using semantic web services for business process management. In *ICEBE '05: Proceedings of the IEEE International Conference on e-Business Engineering*, pages 535–540, Washington, DC, USA, 2005. IEEE Computer Society.

[50] D. Karastoyanova, T. van Lessen, F. Leymann, Z. Ma, J. Nitzsche, B. Wetzstein, S. Bhiri, M. Hauswirth, and M. Zaremba. A reference architecture for semantic business process management systems. In *Multikonferenz Wirtschaftsinformatik*, 2008.

[51] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. Web services choreography description language version 1.0. W3C, November 2005. `http://www.w3.org/TR/ws-cdl-10/`.

[52] R. Khalaf, A. Keller, and F. Leymann. Business processes for web services: principles and applications. *IBM Syst. J.*, 45(2):425–446, 2006.

[53] S. Khoshafian. *Service Oriented Enterprises*. Auerbach Publications, 2007.

[54] A. Kumar and Z. Shan. Algorithms based on pattern analysis for verification and adapter creation for business process composition. In Robert Meersman and Zahir Tari, editors, *OTM Conferences (1)*, volume 5331 of *Lecture Notes in Computer Science*, pages 120–138. Springer, 2008.

[55] H. Lausen, A. Polleres, and D. Roman. Web service modeling ontology - WSMO. W3C, June 2005. `http://www.w3.org/Submission/WSMO`.

[56] J. Lee, Y. Kim, Y. Kim, and B. Moon. Business process integration with web services. *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2005 and First ACIS International Workshop on Self-Assembling Wireless Networks. SNPD/SAWN 2005. Sixth International Conference on*, pages 192–197, May 2005.

[57] F. Leymann and D. Roller. *Production workflow: concepts and techniques*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[58] M. Little. Transactions and web services. *Communications of the ACM*, 46(10):49–54, 2003.

[59] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg. Analyzing interacting BPEL processes. In *Business Process Management*, pages 17–32, 2006.

[60] B. Margolis and J. Sharpe. *SOA for the Business developer: Concepts, BPEL, and SCA*. MC Press, 2007.

[61] A. Martens. *Verteilte Geschftsprozesse - Modellierung und Verifikation mit Hilfe von Web Services*. PhD thesis, WiKu-Verlag Stuttgart, 2004.

[62] A. Martens. Analyzing web service based business processes. In M. Cerioli, editor, *Lecture Notes in Computer Science*, pages 19–33. Springer, 2005.

[63] A. Martens and S. Moser. Diagnosing SCA components using Wombat. In J.L. Dustdar, S.; Fiadeiro and A.P. Sheth, editors, *Lecture Notes in Computer Science*, pages 378–388. Springer, 2006.

[64] A. Martens, S. Moser, A. Gerhardt, and K. Funk. Analyzing compatibility of bpel processes. In *Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006)*, page 147, Guadeloupe, French Caribbean, February 2006. IEEE Computer Society.

[65] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. Mcllraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirim, N. Srinivasan, and K. Sycara. Daml-s/owl-s. W3C, November 2004. `http://www.w3.org/Submission/OWL-S`.

[66] B. Medjahed, B. Benatallah, A. Bouguettaya, A. Ngu, and A. Elmagarmid. Business-to-business interactions: issues and enabling technologies. *The VLDB Journal*, 12(1):59–85, 2003.

[67] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[68] R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[69] L. Momtahan, A. Martin, and A.W. Roscoe. A taxonomy of web services using CSP. *Electronic Notes in Theoretical Computer Science*, 151(2):71–87, 2006.

[70] O. Moser, F. Rosenberg, and S. Dustdar. Viedame - flexible and robust bpel processes through monitoring and adaptation. In *ICSE Companion '08: Companion of the 30th international conference on Software engineering*,

pages 917–918, New York, NY, USA, 2008. ACM.

[71] S. Moser, A. Martens, M. Hbich, and J. Mlle. A hybrid approach for generating compatible ws-bpel partner processes. In J.L. Dustdar, S.; Fiadeiro and A.P. Sheth, editors, *Lecture Notes in Computer Science*, pages 458–464. Springer, 2006.

[72] H.R. Motahari N., B. Benatallah, A. Martens, F. Curbera, and F. Casati. Semi-automated adaptation of service interactions. In C.L. Williamson, M.E. Zurko, P.F. Patel-Schneider, and P.J. Shenoy, editors, *WWW*, pages 993–1002. ACM, 2007.

[73] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[74] A. Norta and P. Grefen. Discovering patterns for inter-organizational business process collaboration. *International Journal of Cooperative Information Systems (IJCIS)*, 16(3):507–544, 2007.

[75] C. Ouyang, E. Verbeek, W.M.P. van der Aalst, S. Breutel, M. Dumas, and A. ter Hofstede. Formal semantics and analysis of control flow in ws-bpel. *Science of Computer Programming*, 67(2-3):162 – 198, 2007.

[76] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007.

[77] M.P. Papazoglou and W.J. Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415, 2007.

[78] R. Passerone, L. de Alfaro, T.A. Henzinger, and A.L. Sangiovanni-Vincentelli. Convertibility verification and converter synthesis: two faces of the same coin. In *ICCAD '02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 132–139, New York, NY, USA, 2002. ACM.

[79] C. Pedrinaci, J. Domingue, C. Brelage, T. van Lessen, D. Karastoyanova, and F. Leymann. Semantic business process management: Scaling up the management of business processes. In *ICSC '08: Proceedings of the 2008 IEEE International Conference on Semantic Computing*, pages 546–553, Washington, DC, USA, 2008. IEEE Computer Society.

[80] J.L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[81] A.W. Roscoe, C.A.R. Hoare, and Richard Bird. *The Theory and Practice of Concurrency*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[82] S.H. Ryu, F. Casati, H. Skogsrud, B. Benatallah, and R. Saint-Paul. Supporting the dynamic evolution of web service protocols in service-oriented architectures. *ACM Transactions on the Web (TWEB)*, 2(2):1–46, 2008.

[83] T. G. J. Schepers, M. E. Iacob, and P. A. T. Van Eck. A lifecycle approach to SOA governance. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 1055–1061, New York, NY, USA, 2008. ACM.

[84] H.W. Schmidt and R.H. Reussner. Generating adapters for concurrent component protocol synchronisation. In *FMOODS '02: Proceedings of the IFIP TC6/WG6.1 Fifth International Conference on Formal Methods for Open Object-Based Distributed Systems V*, pages 213–229, Deventer, The Netherlands, 2002. Kluwer, B.V.

[85] M. Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1996.

[86] M. Stal. Web services: beyond component-based computing. *Communications of the ACM*, 45(10):71–76, 2002.

[87] R. Studer, S. Grimm, and A. Abecker. *Semantic Web Services: Concepts, Technologies, and Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[88] J. Su, T. Bultan, X. Fu, and X. Zhao. Towards a theory of web service choreographies. In M. Dumas and R. Heckel, editors, *WS-FM 2007*, volume 4937 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2008.

[89] W.M.P. van der Aalst, M. Beisiegel, K.M. van Hee, D. König, and C. Stahl. A SOA-based architecture framework. In *The Role of Business Processes in Service Oriented Architectures*, 2006.

[90] W.M.P. van der Aalst and K.B. Lassen. Translating unstructured workflow processes to readable bpel: Theory and implementation. *Information and Software Technology*, 50(3):131–159, 2008.

[91] W.M.P. van der Aalst and Arthur H. M. ter Hofstede. Yawl: yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

[92] W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, 2002.

[93] J. Vonk and P. Grefen. Cross-organizational transaction support for e-services in virtual enterprises. *Distributed and Parallel Databases*, 14(2):137–172, 2003.

[94] J. Vonk, T. Wang, and P. Grefen. A dual view to facilitate transactional qos. In *IEEE International Workshops on Enabling Technologies (WETICE)*, pages 381–383. IEEE Computer Society, 2007.

[95] G. Vossen. From processes via workflows to services: An overview. *Journal of Integrated Design & Process Science*, 10(4):3–11, 2006.

[96] K. Wang, M. Dumas, C. Ouyang, and J. Vayssiere. The service adaptation machine. In *Proceedings ECOWS '08*, pages 145–154. IEEE Computer Society, 2008.

[97] T. Wang, J. Vonk, B. Kratz, and P. Grefen. A survey on the history of transaction management: from flat to grid transactions. *Distributed and Parallel Databases*, 23(3):235–270, June 2008.

[98] S.A. White. Business process modeling notation bpmn 1.1. OMG, January 2008. `http://www.bpmn.org`.

[99] A. Wombacher, P. Fankhauser, and E. Neuhold. Transforming BPEL into annotated deterministic Finite State Automata for service discovery. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services*, page 316, Washington, DC, USA, 2004. IEEE Computer Society.

[100] D.M. Yellin and R.E. Strom. Protocol specifications and component adaptors. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(2):292–333, 1997.

[101] W.L. Yeung. CSP-based verification for Web service Orchestration and Choreography. *Simulation*, 83(1):65–74, 2007.