ARCHITECTURE SUPPORT FOR FLEXIBLE BUSINESS CHAIN INTEGRATION USING PROTOCOL ADAPTORS

RICARDO SEGUEL and RIK ESHUIS and PAUL GREFEN

Eindhoven University of Technology, The Netherlands

Business chains increasingly rely on the dynamic integration of business processes of different partners. The interaction constraints that result from the business processes are captured in business protocols. Since the business protocols of each partner support its own way of working, the business protocols can easily mismatch, which hinders organizations from forming a business chain. Such mismatches can be resolved by protocol adaptors. In this paper, we show how protocol adaptors can be used to enable the flexible formation of business chains. For different types of business chains, we present formation cases that describe which partners are responsible for the construction and operation of protocol adaptors. Next, we present for each formation case an accompanying concrete software architecture that realizes the case. The presented software architectures support the flexible formation of business chains and use protocol adaptation as a key component. We show the feasibility of the approach by discussing a prototype implementation, which we apply to a case study from the healthcare domain.

Keywords: Inter-organizational information systems; business chain; dynamic process outsourcing; interacting services; service adaptation.

1. Introduction

In highly competitive markets, the production of complex products and services involves a number of autonomous organizations that collaborate in business chains. Shorter life cycles for these products and services create the need for agile business chains, which are set up in a just-in-time fashion. An enabling technology for agile business chains is dynamic process outsourcing, which allows an organization to outsource a part of its business process, for instance order fulfillment, to a partner that is selected at the last possible moment. The partner offers its business process part as a service to the outsourcing party. The outsourcing relation can be established by advertising on marketplaces or by searching a repository of requests or offers by organizations.

Once the organizations have established a B2B relationship using dynamic process outsourcing, they can collaborate by invoking functionalities from each other according to their business protocols in their public process view. Each public process view abstracts an underlying private business processes that is executed by the owning organization. Process views are realized by interacting services, which encapsulate the business protocols and enact the global process by exchanging messages.
In existing dynamic process outsourcing approaches, the tacit assumption is made that interacting protocols are compatible. That is, the protocols always interact properly and each sent message can be received and processed by the other party, and thus no deadlocks occur. However, this assumption is not very realistic in domains where no standards have been generally accepted, since in practice each organization has its own protocol that specifies its own way of working. Two collaborating organizations may easily have incompatible protocols. A complicating factor is that the underlying business processes are frequently implemented by legacy information systems that cannot be modified easily to repair incompatibilities.

Since organizations collaborate in a just-in-time fashion, incompatible business protocols hinder the swift and flexible formation of business chains. In modern markets, the globalization of business, shortened time-to-market requirements and increased market transparency are fundamental. Thus, for organizations to remain competitive and collaborate in modern markets, it is essential that business chains can be established in a swift and flexible way.

The goal of this paper is to identify how protocol adaptation can be used to support the quick and flexible formation of business chains, resolving protocol incompatibilities between partner organizations in a semi-automated way. A protocol adaptor ensures that the interaction between two incompatible protocols proceeds properly by intercepting, restructuring and reordering messages such that they are delivered to the receiving side in the format and order that this side expects. Protocol adaptation resolves behavioral and interface mismatches, either in an integrated way or separately. A behavioral mismatch occurs if two interacting services reach a deadlock, each service waiting for the other to send a message. An interface mismatch is due to differences in the formats and specifications of exchanged messages. Such a mismatch can be resolved in a semi-automated way using schema mapping and transformation tools.

We distinguish three types of business chains: supply chains, demand chains and hybrid demand/supply chains. We analyze in Section 2 for each type of business chain which partner in the chain should use protocol adaptors to enable the swift and flexible formation of different chains. Next, we present for each flexible chain formation case a supporting software architecture. The presented software architectures extend and integrate two software architectures from the literature that support the configuration and enactment of supply and demand chain networks. The extensions use business protocol adaptation as a key component to support the flexible formation and enactment of business chains. However, the proposed extensions can be also be applied to other existing software architectures for chain formation.

The main contribution of this paper is a well-structured set of abstract software architectures for flexible business chain formation and enactment. The architectures in this set are based on one template architecture and each support one of the main
business chain paradigms: supply chain, demand chain, and hybrid supply/demand chain. Each architecture shows the configuration of basic functional business process management modules across a chain. In each configuration, protocol adaptation is the main mechanism to achieve flexibility in dealing with incompatible business protocols. We show that the responsibility for support of flexibility depends on the business chain paradigm. We show the feasibility of the approach by a prototype implementation using advanced Web technology. We show the usability of the approach by the application of the prototype in a real-world scenario in the healthcare domain.

The remainder of this paper is organized as follows. Section 2 presents the three business chain types and analyzes for each chain type where and how to use protocol adaptation for flexible formation of chains. Section 3 presents an abstract framework software architecture that we specialize in three concrete software architectures that support the three chain formation cases. Section 4 presents a prototype implementation of the framework architecture, which we apply to a case study from the domain of healthcare. Section 5 discusses related work. Finally, Section 6 presents the conclusions and future work.

2. Flexible Formation of Business Chains

In a business chain, different partners collaborate by performing their internal business processes while exchanging messages with each other. Partner organizations in a chain have business protocols specified in public process views that abstract from private, internal business processes. We illustrate the need for protocol adaptation in the formation of business chains using the example in Fig. 1: even though the partner agree on the format of the exchanged messages (indicated with the arrows), there are behavioral mismatches. An example mismatch is that the consumer W sends a product order message while the provider X expects a message containing delivery details. To enable the formation of a business chain, companies W, X and Y can use protocol adaptation to resolve mismatches. In the example, two protocol adaptors are required that resolve the behavioral mismatches between W and X and between Y and Z.

In theory a trusted third party could be responsible for constructing and enacting protocol adaptors, but introducing such a party in the chain would make the design of the chain more complex and costly. Instead, we envision that partners in the chain are themselves responsible for generating and enacting protocol adaptors.

To analyze which partner is responsible for a protocol adaptor, we use the concept of customer order decoupling point (CODP)\[24,25\] which is the point in the chain where the goods are linked to a specific customer order: downstream of the CODP all organizations produce to order and upstream of the CODP all organizations produce to stock. In a supply chain, the CODP is located at a consumer while in a demand chain the CODP is located at the provider at the start of the chain. In a hybrid demand/supply chain, the CODP is located at a partner in the middle
of the chain, thus dividing the chain in a demand chain downstream the CODP towards the customers and a supply chain upstream the CODP. Figure 1 shows a hybrid demand/supply chain in which the CODP is located at X.

For each chain type, mismatches between interacting business protocols can prevent the formation of business chains. Protocol adaptors can resolve these mismatches. Table 1 analyzes the three different formation cases, which we now explain in detail. The sub-columns BP to Provider, BP to Consumer and BP to 2nd-tier Provider represent the business protocol shared by the services at the public view, and thus they can be either defined by the service (Defined) or compatible with the partner (Compatible/Incompatible). Check marks (√) in the sub-columns Adaptor ②–④ indicate the party responsible for constructing the adaptor, motivated next.

In a supply chain formation case, the service provider defines its business protocol in conformance with market standards like SCOR[20] or eTOM[21]. Consequently, the provider uses standard protocols. If the service consumer has a business protocol that is not compatible with the standard business protocol of the provider, it cannot force the service provider to change the standard. Therefore, the service consumer has to build a protocol adaptor to resolve mismatches with the service provider.
provider (column c in Table 1). If the service provider and 2nd-tier provider operate as supply chain, by similar reasoning the provider is responsible for the adaptor; see column c in Table 1. This way, the number of potential service consumers is increased and the market becomes more competitive although it is regulated for market standards.

In a demand chain formation case, the service consumer defines its business protocol according to the customer business requirements. If the protocol of the service provider is not compatible, the service provider has to construct a protocol adaptor to resolve the mismatches since it cannot force the service consumer to change the requested protocol (column b in Table 1). This way, the number of potential partners is increased, leading to more competitive marketplaces since the selection of partners will be based on both protocol compatibility and other matching characteristics. Since the provider has the business advantage of increased competitiveness, this party should be responsible for using the adaptor.

In a hybrid demand/supply chain formation case, the service consumer and service provider form a demand chain while the service (1st-tier) provider and the 2nd-tier provider form a supply chain. For both subchains, we adopt the reasoning of the previous cases. The service (1st-tier) provider is always responsible for constructing an adaptor to resolve mismatches with the service consumer; see column b in Table 1. On the other hand, the service (1st-tier) provider is always responsible to build an adaptor to resolve mismatches with the 2nd-tier provider; see column c Table 1.

Figure 2 shows the use of adaptors for the example shown in Fig. 1. As in Fig. 1 arrows indicate message sending. The example in Fig. 1 is a hybrid demand and supply chain. Since companies W and X operate in demand chain mode, company X constructs a protocol adaptor to resolve the mismatches with the company W, using one of the existing adaptation methods. Companies X and Y operate in supply chain mode, and therefore company X builds a protocol adaptor.

In both cases, the constructed protocol adaptor is positioned in the public process view. Alternatively, it is possible to specify the adaptors at the private level instead of the public level, and use a public level protocols that mirrors the protocol of the other side. In that case adaptation of a protocol in the private view

<table>
<thead>
<tr>
<th>Cases</th>
<th>Consumer BP to Provider Adaptor (a)</th>
<th>Provider BP to Consumer Adaptor (b)</th>
<th>Supplier BP to 2nd-tier Provider Adaptor (c)</th>
<th>2nd-tier Provider BP to Provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply chain</td>
<td>Incompatible ✓</td>
<td>Defined</td>
<td>Incompatible ✓</td>
<td>Defined</td>
</tr>
<tr>
<td>Demand chain</td>
<td>Defined</td>
<td>✓ Incompatible ✓</td>
<td>Defined ✓</td>
<td>Incompatible ✓</td>
</tr>
<tr>
<td>Hybrid chain</td>
<td>Defined</td>
<td>✓ Incompatible ✓</td>
<td>Incompatible ✓</td>
<td>Defined</td>
</tr>
</tbody>
</table>

Table 1. Flexible Business Chain Formation Cases
is hidden from the counterpart service, which observes a compatible protocol with which it is interacting. So there is no third protocol interacting in the middle at the public level. This alternative does not impact the analysis results of this section, and is therefore not shown.

3. Software Architectures for Flexible Business Chain Formation

In this section, we design three software architectures that support the three flexible chain formation cases discussed in Section 2. Protocol adaptation is a key element in these architectures. To connect to the existing research in the area of chain formation, we decided to take two concrete architectures, one developed for the formation of supply chains and one developed for the formation of demand chains as starting point. For the formation of hybrid demand/supply chains, no software architecture has been proposed in the literature to the best of our knowledge. In this section, we propose a new architecture for the formation of hybrid demand/supply chains.

First, we present a framework architecture that generalizes the two chosen architectures for the formation of supply chains and demand chains. The three concrete architectures developed in the sequel are specializations of the framework architecture. Having a common framework architecture helps to identify common elements between different architectures and has been instrumental in developing the new hybrid architecture. To enable the flexible formation of business chains, each architecture contains one or more protocol adaptation components.

For each concrete architecture, we explain the case in which the protocol adap-
tors are deployed at the public view in front of the original protocols. We also describe the use of a trusted third party to build the adaptors. If protocol adaptors are used at the private view, the architectures change slightly, as we explain elsewhere.

3.1. Framework architecture

In Fig. 3, we show the architecture of a partner organization at a high level of abstraction, which is similar for a service consumer and a service provider. We divide the components into two groups: design-time and run-time components. The arrows indicate the information flows from the design-time components to the run-time components to specify that the chain is configured and then executed. The ‘reverse’ dotted arrow indicates that the chain can be reconfigured when the business protocols are being executed, stopping the execution to configure the chain again.

The team support component is responsible for the selection of a team of business organizations that will execute an inter-organizational business process. This component searches for partners in the marketplace to outsource non-critical parts of a business process to reach a business goal. Each member of a constructed process execution team is a business organization that executes a local process that is a sub-process in the overall, inter-organizational business process. In detail, the goal support component sets the business goal that must be met by the partners in the business chain. The team formation component selects a partner from the marketplace for process outsourcing according to certain matching characteristics and helps to establish the outsourcing relationship. The team support component can have various levels of automation: from fully automated (like in the CrossFlow approach) to semi-automated (like in the CrossWork approach). Automated support in this component is required to reach the levels of efficiency to make flexible chain formation indeed possible.

The process design component checks the partner business protocols, resolves incompatibilities and couples the protocols for later execution. In detail, the process formation component checks the business protocols of the partners to identify possible incompatibilities when they come to collaborate in the outsourcing of the process. If the protocols are incompatible, the adaptor factory generates the protocol adaptor and the protocols are coupled by the process formation component, containing the details for the correct outsourcing of the process.

The process enactment component supports the run-time execution of the protocols generated at design-time. The protocols are deployed in the deployment engine component and executed in the enactment engine component.

The configuration of a demand chain starts with the goal support component, since the service consumer sets the business goal (e.g. produce a laptop) and the business protocol that the service provider must meet for the outsourcing of the processes and the formation of the chain. The configuration of a supply chain starts with the team formation component, since the service provider has a standard
business protocol to accomplish certain business goal (e.g. deliver a book) that the service consumer must meet. The ‘reverse’ dotted arrows indicate the information flows for the reconfiguration of the chain at run-time. The execution of the business protocols is stopped and the configuration of the chain starts again at the design-time components for a supply or demand chain.

### 3.2. Flexible supply chain formation

The CrossFlow architecture was developed to support the configuration of a supply chain, using dynamic process outsourcing. In the CrossFlow business scenario, the service consumer outsources a non-core part of its business protocol to a service provider. The service consumer takes the decision of outsourcing during the execution of its business protocol, and thus it selects the provider at the last possible moment. Thus, the basic CrossFlow architecture is mainly focused on run-time supply chain configuration.

We extend the CrossFlow architecture to support the flexible configuration of a supply chain at design-time and at run-time. Figure 4 shows the basic architecture (white boxes) plus the extension (gray boxes). We have developed an exact mapping that relates the architecture to the framework architecture developed in the previous section.

At design-time, the extended CrossFlow architecture is triggered by the consumer that selects the service provider from the marketplace to outsource its protocol. Then, the provider sends its standard business protocol to the consumer that
checks the compatibility with its protocol. If the business protocols are incompatible, then the service consumer constructs a protocol adaptor. Next, the service consumer couples the outsourced protocol part with its business protocol in the composition module for later deployment in the enactment module. Then, the consumer deploys the coupled business protocol in the protocol enactment module and the adaptor in the adaptor enactment module. Finally, the service provider deploys its business protocol in the enactment module after it sent the protocol definition.

Similarly, the service provider can outsource part of its protocol by selecting 2nd-tier providers from the marketplace. Then, the 2nd-tier providers send their standard business protocol to the provider that checks the compatibility with its protocol. If the protocols are incompatible, then the provider adaptor factory module generates the protocol adaptor. Then, the provider couples the outsourced protocol part with its business protocol in the composition module. Next, the provider deploys the coupled protocol and the adaptor in the enactment modules while the 2nd-tier providers deploy their protocols too. This way, the protocols can be later executed by the service consumer and provider, enacting the supply chain with the 2nd-tier providers.

At run-time, the extended CrossFlow architecture supports the reconfiguration of a supply chain when the consumer business protocol is being enacted. This is illustrated with the ‘reverse’ dotted arrow from the protocol enactment module to the
partner selection module in Fig. 4. Therefore, the consumer stops the business protocol execution to configure the supply chain (at design-time) to later continue the enactment of the business protocols. Then, the service provider can also configure a supply chain with 2nd-tier providers when its protocol is being enacted.

Note that if the service provider acts as integrator only, then the adaptation is only needed at the service consumer. The service provider pre-selects the 2nd-tier providers before it is selected by the consumer. Once the provider is selected, it sends the standard protocol to the consumer. The service provider is protocol compatible with the 2nd-tier providers since they interact only to synchronize the control flow of the business protocol. On the other hand, the consumer protocol interacts with the standard business protocol, which can require adaptation.

Technically the business protocols and the adaptor can be enacted using the same enactment engine or two different enactment engines. The protocol adaptor enacted at the service consumer interacts with the business protocols of the consumer and provider.

3.3. Flexible demand chain formation

The CrossWork architecture was developed to support the dynamic formation and enactment of a demand chain. In the CrossWork business scenario, the service consumer is an Original Equipment Manufacturer (OEM) organization. The OEM defines a certain goal or solution objective that is sent to the service provider. Then, the provider forms the chain by selecting the 2nd-tier providers from the marketplace to meet the goal. The provider composes a global business protocol with the local protocols of the 2nd-tier providers to coordinate them. Then, the provider enacts the global process that enacts the local protocols in the 2nd-tier providers to later send the result back to the OEM.

We extend the CrossWork architecture to increase the search space of potential partners. The original architecture only supports potential partners that have compatible protocols, thereby discarding partners with incompatible protocols, but otherwise matching characteristics. The extension allows to also consider these partners. It enables the flexible formation of a demand chain, by adding the necessary architecture components to support protocol adaptation. Figure 5 shows the CrossWork architecture (white boxes) plus the extension (gray boxes). An exact mapping that relates the architecture to the framework architecture developed in the previous section is discussed elsewhere.

The extended CrossWork architecture enables the flexible formation of a demand chain at design-time and at run-time.

At design-time, the extended CrossWork architecture is triggered by the service consumer that defines the solution objective in the goal support module according to the customer business requirements. The consumer defines its business protocol according to the goal and selects a service provider from the marketplace that meets the goal. The consumer sends the goal definition and its business protocol to the service provider.
Then, the service provider checks the compatibility between its business protocol and the consumer protocol. If the protocols are incompatible, the adaptor factory module constructs a protocol adaptor to resolve mismatches. The adaptor factory implements the adaptation method for tightly and loosely coupled interacting services. Then, the provider deploys the business protocol in the protocol enactment module and the protocol adaptor in the adaptor enactment module. Similarly, the service consumer deploys its business protocol in the protocol enactment module.

Next, the service provider decomposes the goal into a required set of protocols (component services) in the goal support module. Then, the team formation module finds the 2nd-tier providers in the marketplace according to the set of protocols. Next, the composition module composes the set of protocols into a global business protocol.

Note that the 2nd-tier providers are not only selected by protocol compatibility. It means that while there are parts of the global protocol that are compatible with the 2nd-tier providers, there are other parts that are incompatible with the 2nd-tier provider protocols. This way, the 2nd-tier providers check the compatibility between their business protocols and the set of protocols that compose the global protocol. If they are incompatible, the adaptor factory at the 2nd-tier providers generates a protocol adaptor. Next, each 2nd-tier provider deploys the business protocol in the
protocol enactment module and the adaptor in the adaptor enactment module.

Finally, the service provider deploys the global protocol in the protocol enactment module. This way, the protocols can be later executed by the service consumer and provider, enacting the demand chain with the 2nd-tier providers.

At run-time, the extended CrossWork architecture enables the reconfiguration of the demand chain when the consumer business protocol is being enacted. This is illustrated with the ‘reverse’ dotted arrow from the protocol enactment module to the goal support module in Fig. 5. This way, the consumer stops the business protocol execution to configure the demand chain (at design-time) to later continue the enactment of the business protocols. Similarly, the service provider can configure the demand chain with 2nd-tier providers when its protocol is being executed.

Note that if the service provider acts as integrator only, then the adaptation is only needed at the 2nd-tier provider. The service provider uses the protocol sent by the consumer as global business protocol. Then, the service provider is protocol compatible with the service consumer since they interact only to synchronize the control flow of the global business protocol. The global business protocol interacts with the 2nd-tier providers, which can require adaptation.

In the basic CrossWork architecture, if the global protocol cannot be composed then the system backs up to the team formation or even to the goal support module to correct it. However, in the extended architecture, these backtrack steps are needed only if the adaptor factory module at the 2nd-tier provider indicates the mismatch cannot be resolved and no adaptor can be constructed.

As in the extended CrossFlow architecture, the business protocols and the adaptor can be enacted using the same enactment engine or two different enactment engines.

3.4. Flexible hybrid demand/supply chain formation

The architecture to support the flexible configuration of a hybrid demand/supply chain has to enable the flexible formation of a demand chain between the service consumer and service provider. Moreover, it has to enable the flexible formation of a supply chain between the service provider and 2nd-tier providers. Therefore, we define the architecture as a combination of the CrossWork (demand chain) and CrossFlow (supply chain) architectures. The architecture, shown in Fig. 6, supports the configuration of a chain at design-time and at run-time. As before, we have defined an exact mapping that relates the architecture to the framework architecture.

At design-time, the architecture is triggered by the service consumer that configures a demand chain with the provider. The consumer defines the solution objective in the goal support module according to the customer business requirements. Then, the consumer defines its business protocol according to the goal and selects the service provider from the marketplace that meets the goal. Then, the consumer sends the goal definition and its business protocol to the service provider.

Next, the service provider checks the compatibility between its business protocol
and the consumer protocol. If the protocols are incompatible, the adaptor factory module constructs a protocol adaptor to resolve mismatches. Then, the provider deploys the business protocol in the protocol enactment module and the protocol adaptor in the adaptor enactment module while the service consumer deploys its business protocol in the protocol enactment module.

At this stage, the service provider configures a supply chain with 2nd-tier providers. Next, the service provider decomposes the goal into a required set of protocols (component services) in the goal support module. Then, the team formation module finds the 2nd-tier providers in the marketplace according to the set of protocols. Next, the composition module composes the set of protocols into a global business protocol.

Then, the 2nd-tier providers send their standard business protocol to the provider that checks the compatibility with its protocol. If the protocols are incompatible, then the provider adaptor factory module generates the corresponding protocol adaptors. Finally, the service provider deploys the global protocol and the adaptors in the enactment modules while each 2nd-tier provider deploys its protocol in the enactment module too. This way, the protocols can be later executed by the services by enacting the demand chain between the consumer and provider and the supply chain between the provider and 2nd-tier providers.

At run-time, the extended architecture supports the reconfiguration of the hy-
brid demand/supply chain when the consumer and provider protocols are being enacted. This is illustrated with the ‘reverse’ dotted arrow in Fig. 4. The consumer stops the business protocol execution to configure the demand chain (at design-time) to later continue the enactment of the business protocols. Similarly, the service provider stops the protocol execution to configure the supply chain (at design-time) with the 2nd-tier providers to later continue with the protocol enactment.

Note that if the service provider acts as integrator only, then the adaptation is needed for compatibility with either the 2nd-tier providers or the service consumer. In both cases the adaptor is constructed and deployed by the service provider.

In the first case, like in the extended CrossWork case, the service provider uses the protocol sent by the consumer as global business protocol. Then, the provider and consumer protocols are compatible since they interact to only synchronize the control flow. Thus, the global business protocol parts interact with the 2nd-tier providers, which can need adaptation.

In the second case, like in the extended CrossFlow case, the service provider pre-selects the 2nd-tier providers before it is selected by the consumer. Then, the provider has to generate an adaptor if the consumer and provider protocols are incompatible. The service provider and the 2nd-tier providers are compatible since they interact to only synchronize the control flow of the global business protocol parts.

3.5. Third-party adaptor factory

The architectures that we have defined for flexible formation of business chains support the participation of a trusted third-party that provides adaptation as a service (AaaS). The third-party is an adaptor factory that constructs and enacts protocol adaptors to support the flexible configuration of business chains. This way, the service consumer, service provider or 2nd-tier providers can buy the adaptation service from the trusted third-party, and thus they do not need to add the adaptor factory module in their architectures themselves.

The use of a third party by one of the partners does not change the responsibility of the different partners for performing adaptation as described before in Section 2. Related, the participation of the third-party does not cause conceptual changes to the architectures defined previously, but additional technology is needed to ensure quality of services and security, which are outside the scope of this paper.

3.6. Extension of other architectures

To further assess the feasibility of the proposed extensions, we have performed an extensive literature survey of architectures for the formation of business chains. Section 5 discusses this related work in detail. All the considered architectures contain at a high-level of abstraction one or more of the three main components in Fig. 3: a team support component to search for partners and establish the outsourcing relationship between the partners; a process design
component to configure the partner business protocols; and a process enactment component to execute the business protocols in the business chain. By mapping these existing architectures to the framework architecture as shown in Fig. 3, we have identified possible extensions for them relating to protocol adaptation. The extended systems add flexibility to the formation of a business chain.

3.7. Discussion

The proposed architectures assume that specifications of executed business processes are explicitly represented as artifacts that can be exchanged between modules of the software systems that embody the architectures. Thus, the architectures are not applicable to systems that manage hardcoded processes such as ERP systems. More in general, target systems should be sufficiently open such that they support configurable interactions between different systems components in order to realize the architecture.

Next, the architectures assume that the process management components are sufficiently open such that they allow that the processes they manage are partly controlled by another component, i.e., an adaptor. As we discuss in the next section, current industry standards such as BPEL and related software implementations of these standards do not satisfy this assumption. Note however, that interaction between multiple controllers (process engines) was already included two decades ago in the reference framework of the Workflow Management Coalition (specifically, in Interface 4 of the framework) - one might therefore consider the fulfillment of the assumption a “pending issue”. Consequently, the architectures in this paper are not directly realizable using state of the art technology currently in use in business practice. However, we also discuss in the next section that several research prototypes support such an advanced control of processes, and we discuss a dedicated research prototype that we have developed and applied in a case study.

An important issue in the realization of the architectures described in this paper is the level of automation in adaptation. Depending on the usage scenario, adaptation can be fully automatic or semi-automatic. Even in the semi-automatic case, human work is typically limited to choosing between automatically generated options. Thus, the time span required for the manual configuration of chains is (very) small when compared to the time span required for the execution of instances of processes in these chains (in the used healthcare process, for example, the latter is typically in the order of multiple days). Consequently, the architecture supports just-in-time scenarios for supply chains, but not for say financial processes which have a very short time span for executing process instances.

4. Implementation

To show the feasibility of the approach presented in this paper, we have built a prototype of a supply chain support system providing chain flexibility by the use of protocol adaptation. This section discusses the prototype as well as an example
inter-organizational process from the domain of healthcare to which we have applied the prototype.

The structure of this section is as follows. Below, we first analyze existing technology for the embodiment of our architectures. Next, we discuss how to extend this technology to fit our purposes. We show how the extended technology has been applied in the realization of our prototype system. Next, we show how this prototype system is used for the support of flexible supply chains in the healthcare domain. We conclude this section with a discussion of lessons learned.

4.1. Current technologies and their limitations

We first evaluate to what extent current technologies can be used to implement the concrete software architectures that we have defined in Section 3. BPEL and WSDL are the de-facto standard languages for realizing business processes using service-oriented technology. BPEL is a service composition language that is enacted in a centralized manner, i.e., a BPEL protocol orchestrates the autonomous execution of the composed atomic services. A composite BPEL service is wrapped in a WSDL service that describes the function calls and formats to execute the BPEL composition. The WSDL service is invoked as a black-box: details of the execution of the internal BPEL protocol are hidden from the caller.

However, in this paper we consider interacting services whose message interactions are governed by business protocols. Such stateful protocols cannot be encapsulated in WSDL services, which are stateless. Hence, we have evaluated different technology solutions aimed at realizing interacting white-box services.

(1) Business Process Web Service (BP-WS) is a wrapper component that provides ports to obtain the specification of the BPEL protocol (public view), control the execution of the protocol being enacted (start, stop, pause, restart, etc.) and monitor its execution. To implement a BP-WS service, the wrapper, ports and functions have to be manually hard-coded, since there is no standard and tool support to generate the artifacts (semi-)automatically. Consequently, this approach is not usable for the (semi-)automatic generation of protocol synchronization components.

(2) Service Component Architecture (SCA) is a standard that provides a wired model between components implemented in one of many languages, e.g., Java or BPEL. SCA defines the support for BPEL components, but there is no technology available to implement a wired model with BPEL components that interact through communicating messages and use fine grained synchronization.

(3) Web Services Choreography Description Language (WS-CDL) is a standard that provides an interaction model for the choreography of services. However, it does not offer a good integration with BPEL to implement interacting services. WS-CDL relies on the WSDL descriptions of the services and so any change in a WSDL file requires a reconfiguration of the WS-CDL choreography description.
(4) BPEL4Chor\textsuperscript{50} extends BPEL by adding choreography-specific concepts. It provides a choreography model on top of BPEL and better integration than WS-CDL, i.e., BPEL protocols can be generated out of BPEL4Chor choreographies. A BPEL4Chor model defines a participant topology to specify the participant types, references and message links; a participant behavior description that specify the control flow and data flow dependencies between participants; and a participant grounding to add the technical configuration of the choreography to link WSDL with the message formats and types. The limitation of BPEL4Chor is that the specification is not broadly adopted in industry and there is only little support and documentation in existing development tools.

(5) BPELgold\textsuperscript{51} is an interaction language that extends BPEL4Chor to provide and interaction model on top of it. The interaction model is defined using an abstract BPEL process specification to model the behavior of all the participants instead of modeling each participant’s behavior separately. BPELgold reuses the participant topology description and the participant grounding of BPEL4Chor and adds the interaction model. Like BPEL4Chor, BPELgold lacks support in existing development tools and has very little documentation.

We selected BPEL4Chor and BPELgold as a basis for the implementation of the prototype system, since they provide choreography functionality and are tightly integrated with BPEL. Even though these technologies are not yet industry-fit, their use avoids the realization of another basic technology layer with similar functionality.

As we have discussed in this paper, interacting services that have incompatible protocols cannot collaborate. So the adaptation component defined in the concrete software architectures (cf. Section\textsuperscript{3}) must generate the adaptor to resolve the mismatches. In the next subsection, we show how the adaptation component extends the current tool chain of BPELgold to implement interacting services in the prototype system.

4.2. Extending current technologies for adapting interacting services

Figure\textsuperscript{7} shows the BPELgold model chain.\textsuperscript{52} BPELgold models are generated manually. A BPELgold model includes an interaction description and the artifacts of a BPEL4Chor model. The interaction description corresponds to modified abstract BPEL models of the interacting services that restricts the use of input/output variables and removes the partnerLink, PortType and operation elements. A BPEL4Chor model can be transformed to automatically generate the abstract BPEL model of the participants of the choreography.\textsuperscript{53} The grounding definition contains the details of the message formats and types described in the WSDL definitions. The abstract BPEL models are refined manually with the WSDL definitions to generate the Executable BPEL models. The interacting services are executed in the BPEL engines, according to the global interaction model (BPELgold).
Since adaptation of the interacting services is not addressed in the original model chain for BPELgold, we have extended the tool chain by adding minimal adaptors (gray box) which can be automatically generated by analyzing abstract BPEL protocols. We have developed a tool, explained in the next subsection, that automatically generates a BPEL adaptor to resolve incompatibilities between the interacting services. However, any other adaptation tool that generates BPEL adaptors can be used instead in the embodiment of our architecture. The adaptor is encoded as an abstract BPEL model, which represents an orchestration model. Then, the adaptor must be added as a new participant to the choreography. The adaptor is added to the interaction model to specify how the messages are communicated with each other. The model flow in Fig. 7 is top-down and describes the implementation of the interaction model and executable processes at design-time.

4.3. Prototype

Figure 8 shows the high-level structure of the prototype system as an embodiment of the framework architecture shown in Fig. 3. The prototype can be extended to implement the architectures in Fig. 4, 5, and 6. The prototype uses BPEL to specify and enact business protocols. A BPEL service is wrapped in a WSDL service that describes the function calls and formats to execute the composition.

The design-time part of the prototype system is based on Eclipse. The prototype system realizes the Process Design module, which supports the definition of BPELgold, BPEL4Chor, BPEL and WSDL models, and includes an adaptation tool that we developed in earlier work. We have omitted an implementation
of the Team Support modules shown in Fig. 8 since the matching algorithm and partner selection tool are not required to show the feasibility of the adaptation of interacting services. The goal definition module can be implemented with JADE and Eclipse to support high-level, knowledge based reasoning, as has been done in the CrossWork system. The matching and selection tool has also been implemented in the CrossWork system.

The CASmix system developed by Kopp et al. is used as basis for implementing the run-time part of prototype system, i.e. the Process Enactment module. CASmix is a choreography-aware enterprise service bus that supports the execution of interacting BPEL protocols specified in BPELGold. CASmix is based on the enterprise service bus Apache ServiceMix, the messaging server Apache ActiveMQ and the BPEL engine Apache ODE. In CASmix, ODE and ServiceMix are extended with custom Java Business Integration (JBI) components to capture the state of a choreography. The CASmix prototype enables the execution of two interacting BPEL services, each BPEL service being enacted in a separate ODE engine. If two interacting services reach a deadlock, the choreography has a fault and the system reacts with a fault handling action. By adding our adaptation tool to CASmix, we are able to generate an adaptor at design-time and modify the choreography and interaction models by including the BPEL adaptor protocol as a third service.

In the prototype, adaptors are generated at design-time by analyzing protocols before they are deployed. Alternatively, the adaptor generation can also be trig-
gered at run-time when a deadlock occurs whilst the interacting services are being executed. The run-time adaptation is supported in the framework architecture in Fig. 3 with the dotted ‘reverse’ arrows, but has not been realized in our prototype. Adding this functionality is a straightforward extension, however.

4.4. Healthcare case study
As a case study, we have applied the prototype to an inter-organizational business process in the healthcare domain. We have chosen the healthcare domain for prototyping, because this is a domain with many diverse, inter-organizational business process chains. The diversity of the chains is caused by the many combinations of clinical pathways for complex, multi-party treatments. Growing complexity and demands for efficiency call for automated support for these complex pathways and the execution of the underlying inter-organizational business processes. The chosen case study is based on a tele-radiology process for the acquisition and interpretation of medical scans (X-ray, CT, or MRI scans) of patients. The process involves two parties: a hospital in which a patient is scanned and a laboratory that interprets that scan. The scan is sent by the hospital to the laboratory and the laboratory sends back a report about the scan. The process has been designed in close collaboration with an industrial partner that offers technology support for certain parts of this process.

The tele-radiology process starts when the hospital requests a scan by sending an order, a date and the patient data to the laboratory. Next, the hospital requests an urgent report and waits for the scans and the pre-diagnosis report. The laboratory sends the scans and the pre-diagnosis to the hospital. Then, the hospital receives the urgent report and always requests an extra analysis to be included in the final report. Finally, the hospital receives the final report from the laboratory.

The protocols and their interactions are depicted in Fig. 9. The message exchanged (interactions) between the protocols are represented by dotted arrows. Directions of dotted arrows indicate the send (source) and the receive (target) nodes. Note that message labels are not shown in the figure; they are implied by the names of send and receive actions.

The interacting services cannot collaborate, since the business protocols contain mismatches. Figure 9 shows the interactions that cause deadlock with bold arrows, assuming synchronous communication between the parties. For instance, the hospital wants to send message ScanOrder but the laboratory can only receive message PatientData. Therefore the execution blocks. By letting an adaptor receive message ScanOrder and ScanDate, the next message PatientData can be sent by the hospital and received by the laboratory. Using the approach we developed in earlier work, we can derive a minimal protocol adaptor that only processes those messages that cause the deadlocks. For this example, the generated synchronous adaptor processes eight messages: ScanOrder, ScanDate, UrgentReport, CTScan, MRI Scan, XRayScan, PreDiagnosis and FinalReport. Note that this mismatch is not resolved by
using asynchronous communication; for instance, the hospital can asynchronously send the message sequence **ScanOrder, ScanDate, PatientData** to the laboratory, but the laboratory expects these messages in a different order.

Figures 10 and 11 show the code snippets of the BPEL protocols of the Hospital and the Laboratory respectively. The prototype system contains two ODE engines in which each protocol is deployed. In addition, an adaptor that resolves the mismatches between the Hospital and the Laboratory needs to be generated. We have used a minimal adaptor\textsuperscript{37,35} to resolve the mismatches. A minimal adaptor only adapts the messages of interactions that cause the deadlock.

Figure 12 shows the output by the adaptation tool\textsuperscript{57} that is part of the prototype system (cf. Fig. 8). The figure shows the protocol trees of the hospital (left-side), the laboratory (center) and the synchronous minimal adaptor protocol tree (right-
side). The set of interactions needing adaptation is depicted at the bottom of this figure. The BPEL code is listed in Ref. 49.

Both the abstract BPEL models for the Hospital and the Laboratory and the generated BPEL model of the adaptor are used to define the choreography model in BPEL4Chor. The topology of the choreography is shown in the code snippet in Fig. [13] As a result, the protocol adaptor is added to the choreography. Then, the topology is used for the interaction model in BPELgold. Finally, the interaction model and the choreography are deployed in CASmix.

4.5. Discussion

In this section, we have discussed the realization of a prototype embodiment of the architecture proposed in this paper and an application of this prototype in a practical context. From this experience, we can draw two kinds of lessons learned: one about technology and one about application domains.

When we look at technology, we find that on the one hand that Web-based processes have been around for substantial time now. For example, the BPEL standard was proposed in 2003. On the other hand, however, we find that industry-
strength support for advanced Web-based process aspects, such as choreography and adaptation, still have very limited technical support at the practical level. Consequently, we had to choose a prototype infrastructure as the basis for the implementation of our prototype environment. We observe a severe gap between advanced, inter-organizational BPM technology (which is mainly research-driven) and basic industry-strength support for this.

When we look at application domains, we observe a growing application field for dynamic, inter-organizational business processes. In this section, we have already discussed the healthcare domain, where more and more inter-organizational, patient-oriented business processes need to be deployed for effective and efficient execution of complex medical procedures. The healthcare domain, however, is not yet that standardized that compatible, standard local processes exist. In other words, adaptation will be an important technology in this domain. A similar story holds

Fig. 12. BPEL Adaptor prototype applied in the tele-radiology example case
for the logistics domain, where complex, inter-organizational business processes are required for the control of international, multi-modal transport operations again with lacking standardization at the process level.

5. Related Work

There is a large body of work on protocol adaptation, which targets behavioral and interface mismatches. However, none of these works discuss how adaptation can be used to improve the formation of supply and demand chains, which is the topic of this paper. Note that any of these existing protocol adaptation approaches can be used to realize the actual adaptation of incompatible protocols. This way, our work complements the existing work by showing how it can be applied to enable the flexible formation of business chains.

Though there is substantial related work on inter-organizational workflow collaboration, it does not address adaptation in the architecture definitions. Thus, the proposed architectures do not fully support the flexible formation of business chains. We now explain each work in detail to show how it can be extended, if possible, with the new architecture components we have presented in Figures 4–6.

The architecture defined by Chiu et al supports process views and it is divided in four layers: back-end (OODBMS), workflow (WFMS), internet (E-ADOME) and agents. The workflow layer contains workflow editor (design-time) and workflow enactment (run-time) components. It supports workflow evolution and manual excep-
tion handling to resolve unexpected exceptions at design-time, but no adaptation to resolve protocol mismatches automatically. Moreover, the architecture is defined for an integrator that receives the customer requirements and send them to the service providers, which construct and finally deliver the goods. That work is focused on demand chains and can be extended with the components of the architecture shown in Fig. 5. This way, we extend the architecture of the service provider (integrator) by adding design-time components: protocol compatibility check and adaptor factory, linked to the workflow editor; and a run-time component: adaptor enactment linked to the workflow enactment module. Moreover, the architecture can be extended to support interacting services by adding the components of Fig. 5 for the service consumer and the 2nd-tier provider.

The architecture presented by Liu et al. defines process views for each partner to create a collaboration workflow that represents the integrated supply chain. The architecture extends the work done before by Grefen et al. in the CrossFlow architecture by adding the process view concept for collaboration workflows. It contains a definition module (design-time) for partner selection and protocol composition and an enactment module (run-time) for the protocol execution. However, the architecture neither supports interacting services nor any mechanism to resolve mismatches between partners to flexible create the collaboration workflow to enable the supply chain. Therefore, we can extend this architecture adding the necessary components of the architecture shown in Fig. 5. First, in the definition module we add the protocol definition linked to the partner selection, the protocol compatibility check and the adaptor factory components, all they linked as we show in the figure. Second, we add the adaptor enactment component linked to the protocol enactment component. Third, to enable interacting services we extend the architecture with the service consumer and 2nd-tier provider components as we showed in Fig. 4.

The architecture for workflow integration of Jung et al. is divided in three interfaces (or layers): human, B2B and application. The B2B interface include web service standards for interoperability and message exchange. Like Ref. 38, the approach enables a supply chain by means of public process that is shared by the partners, but no interacting services. Also, the B2B interface contains a workflow engine to enact the public process. Although the authors mention adaptor support as a requirement for B2B integration, it is not included as a component in the B2B layer. The architecture includes a component in the application layer for application-specific adaptors (wrappers) that allow back-end integration. Thus, these kind of adaptors are developed for each type of application. Moreover, the architecture only describes run-time components and usage, but no design-time components. We can extend the architecture adding the design-time components: protocol definition linked to protocol compatibility check that is linked to the adaptor factory; and the run-time component adaptor enactment, linked to the workflow engine; see Fig. 4. This way, to enable interacting services we add also the components in the service consumer and 2nd-tier provider.

The CrossWork architecture enables the dynamic creation of virtual enterprises
by means of peer-to-peer workflow collaboration. The architecture defines a global protocol from the customer requirements like Ref. 13, but it decomposes the protocol in subparts for n-tier providers. The architecture supports process views \cite{8,10,11}. The architecture has a compatibility checking component (at design-time) for provider protocols which are finally discarded if they mismatch, and thus no protocol adaptation component is included to enable more flexibility for the chain formation and enactment (at run-time). We have extended this architecture as we detailed in Section 3.3 and showed in Fig. 5.

The architecture presented by Jiang et al.\cite{16} integrates in a public protocol the process views of partners to collaborate in product development. It supports demand chains. The architecture has process view definition and execution components in an interface layer that is connected to the back-end systems in the workflow layer, but it does not support interacting services. Moreover, the architecture lacks a mechanism to check compatibility between partners (at design-time) and the exception manager module is restricted to only deal with execution failures of the shared protocol (at run-time). Then, adaptation is not included to support flexible formation of business chains. Therefore, to extend this architecture we add the components for the design-time part (see Fig. 5): protocol compatibility check and adaptor factory; and adaptor enactment for the run-time part. Moreover, to support interacting services the architecture has to be extended with the corresponding components for the service consumer and 2nd-tier provider.

Other architectures for B2B collaboration \cite{38,39,40,41} do not support process views, but they have been defined for ad-hoc supply chain integration. Laczana et al.\cite{38} presents a software platform for process based business-to-business electronic commerce to support for networks of small and medium enterprises. The architecture relies on a central workflow engine to control inter-organizational processes that are manually designed, and thus it does not support automatic composition of global processes. Verwijmeren\cite{39} presents a software component architecture to integrate local systems (resource planning, warehouse and transportation) by means of supply chain engine components. However, the interaction between these components is restricted to interfaces developed with the same technology, and thus neither workflow integration nor adaptation are envisioned in the design of the architecture to allow flexibility. The architecture described by\cite{40} includes integration interfaces to support process outsourcing based on rules that define a collaborative flow of messages that are handled by a queue based component. Thus, the architecture neither supports an advanced workflow integration nor adaptation. Finally, Li et al.\cite{41} present an architecture of a platform for process outsourcing that integrates partner systems encapsulated in services that are orchestrated in a collaboration process and executed in a service bus. The architecture neither supports interacting services nor adaptation to flexibly form business chains. This way, none of these architectures\cite{38,39,40,41} can be extended.

Based on this analysis, we conclude that the architectures we present cover most of the essential components of the existing architectures presented in related
work. Moreover, these existing architectures can be extended in a similar way as the architectures extended in Section 3 by adding the necessary components to support interacting services, protocol adaptation and to enable the flexible formation of business chains.

6. Conclusions and Future Work

We have presented three different cases for the flexible formation of business chain types. For each case we have defined the supporting software architecture that considers protocol adaptation component as a key enabler of flexible chain formation. Any existing protocol adaptation approach can be used to realize the actual adaptation. The architectures support adaptation at the public level, but can be easily changed to support adaptation at the private level.

Our approach enables the flexible formation of business chains between organizations that collaborate in a just-in-time fashion to meet a solution objective. Thus, existing business chain relations can be established in a more efficient and agile way. Trading marketplaces can be made more competitive and flexible to enable dynamic collaboration networks. New business chain networks can emerge to enable new business models in different domains, leading to promising applications, products and services.

There are several directions for further work. The approach can be extended to deal with adaptation of running business chains. New business requirements or changing market conditions can force a partner to change its internal business processes, and then the protocols that encapsulate these business processes need to be changed as well. Such a local change can lead to a deadlock if the collaborating partners do not change their protocols. However, a protocol adaptor can compensate such a local change, so by using protocol adaptors the business chain can become more resilient to local protocol changes. In future research we plan to identify which run-time changes can be dealt with locally by using protocol adaptors and which local changes require all partners in the business chain to change their protocols.

Next, the presented architectures can be extended to support monitoring of quality of service features, as for instance captured in Service Level Agreements (SLAs). Through the use of adaptation, the search space for potential partners will significantly increase, thereby opening ways to more diverse collaboration. This increases the need for explicit SLA management.

From a business application point of view, an analysis and classification of application scenarios is an interesting direction for future work. Using such a classification, the applicability of one (or multiple) of the proposed architectures can be concretely assessed. This can be combined with guidelines for the technical embodiment of the functional modules in the architectures.

References

23. Z. Shan, A. Kumar, and P. Grefen. Towards integrated service adaptation - a new
45. R. Chinnici, J. Moreau, A. Ryman, and S. Weerawarana. Web services description
30 R. Seguel, R. Eshuis and P. Grefen