Towards a Theory of Web Service Choreographies

Jianwen Su[1⋆], Tevfik Bultan[1], Xiang Fu[2], and Xiangpeng Zhao[1,3]

[1] University of California at Santa Barbara
[2] Georgia Southwestern University
[3] Peking University, China

Abstract. A fundamental promise of service oriented architecture (SOA) lies in the ease of integrating sharable information, processes, and other resources through interactions among the shared components that are modeled as web services. It is expected that not only the participating services are complex and have observable states, but the number of interacting services may be also large. Prior work on choreographies (conversation protocols) all focuses on specifying how the interacting web services should behave globally. Studies have shown that the relationships between global and local specifications of service interactions could be rather intricate. In this paper, we formulate a framework consisting of logical and implementation levels. We survey and discuss the technical problems and known results concerning service design, analysis and verification in this framework.

1 Introduction

A fundamental principle of Service Oriented Architecture (SOA) is to design and model complex software systems as assemblies of bitesize pieces. The pieces can then be managed and re-used. While the paradigm is promising, there is a serious lack of principles to aid the design of complex systems from the existing pieces, and to help the management of systems, small or large. This paper aims at the former problem and attempts to develop a technical framework on which service design principles can be developed. The framework is based on application needs as well as technical results concerning composite service design, analysis and verification developed in the community.

Two characteristics distinguish service design from distributed system design studied in the past. First, working with abstractions is a necessity rather than a preference. There are many reasons a service provider will not reveal the detailed information concerning the internals of a service. Service design must rely on the abstract description of the needed services. Furthermore, it is often required that an abstraction of the composition is fully developed [24, 3] which can serve as either a design specification or constraints for verification. This high level abstraction is built from the observable actions of participating services but it is different from system traces. Second, as the SOA popularity grows, the number of available services also increases rapidly. It is necessary to automate (or semi-automate) many steps in service design.

In the services computing community, there have been investigations concerning the design, analysis, and verification of service compositions. Most of the prior work

⋆ Supported in part by NSF grants IIS-0415195 and CNS-0613998.

either focuses on (proposed) standards or concerns sophisticated techniques in various aspects. As an important SOA application domain, business applications embrace SOA on one hand, but on the other hand are struggling with the lack of a framework that can address the complete service design cycle [4].

In this paper, we formulate a technical framework that consists of two levels of abstraction: logical and implementation. At the logical level, specifications focus on how participating services should interact with each other, while the implementation level provides abstractions of services. Among other things, the framework solidifies from a formal perspective the differences between WS-CDL [29] and BPEL [6]. We give a survey on existing technical results over this framework.

This paper is organized as follows. Section 2 gives a general discussion on service design approaches. Section 3 surveys the existing choreography models. Section 4 focuses on the key technical problems concerning service design and analysis. Section 5 concludes the paper.

2 SOA and Service Design

In this section, we give a general discussion on service design under the influence of SOA (service oriented architecture). We argue that service design needs two or more levels of abstraction. On the logical level, specifications will focus on interactions among services; on the implementation level, the goal is to allow service executions to satisfy logical specifications.

A fundamental premise of SOA is to structure complex software systems into "bite-size" pieces, which can then be easily managed and reused. Such a framework is a clear departure from the traditional software development approaches aiming at individual software components due to the changes in many phases of the development process [5]. Among many technical issues is design methodology for constructing new services (software systems) from assembling existing services.

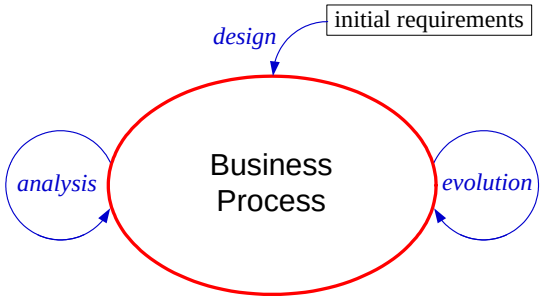


Fig. 1. Life cycle of a business process

To illustrate the issues, consider as an example the life cycle of a business process (to handle, e.g., purchase orders, loan applications, etc.) as shown in Fig. 1. In the design phase, business requirements are used to eventually produce an operational system. The efforts in this phase could involve designs in multiple layers, from high-level conceptual to eventual coding. Automated or semi-automated *design* and *analysis* tools

will provide a significant help in reducing the development time and in improving the quality of design. During the operation phase, business processes, in particular, need to make changes to adapt to the environment better (changes in the market, laws, etc.) and make improvements to achieve business goals. *Evolution* tools could provide support for monitoring the executions, assessing impact of potential changes, and even making the changes.

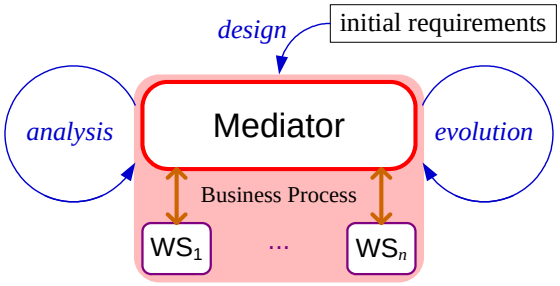


Fig. 2. A mediator process *orchestrates* "component" services

If we focus on the result of the design process from the initial requirements, it is necessary to understand service design methodology. Traditional approaches basically treat the services used as "components" in constructing a new service (or software component). The new service "orchestrates" the component services (Fig. 2). We broadly call such an approach *orchestration*. Typical examples include BPEL [6] and many workflow systems.

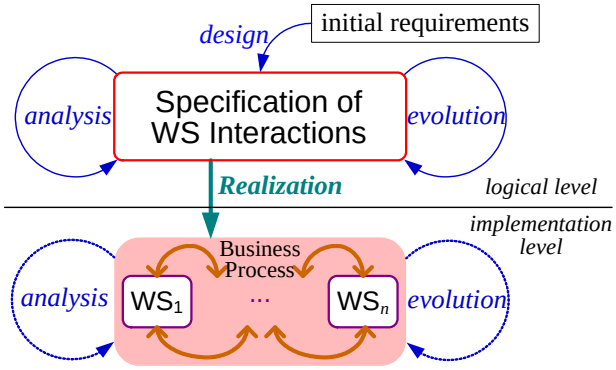


Fig. 3. A *choreography* specifies the interactions between services

There is, however, a new, different methodology called *choreography* that was recently proposed in services computing community [20, 21, 10, 29]. This new approach assumes that the participating services, once "connected", will run on their own with no global intervention; the composite service design is then to specify *how* and *when* the participating services should interact with each other. Fig. 3 depicts this scenario.

Comparing the orchestration and choreography approaches, a mediator is an executable program and a part of the implemented business process. Therefore, analysis, monitoring, and updates will need to be performed on the mediator and the used ser-

vices as a whole. On the contrary, a choreography is merely a logical specification of the observable behaviors by the interacting services at the logical level (Fig. 3). In the ideal case, the choreography completely captures the service behaviors; thus, analysis and updates would only need to be done at the logical level. We believe that a clear separation of logical and implementation levels should be fundamental principle that would allow us to separate and localize concerns and make them independent in developing better systems. The principle is reminiscent of a similar principle in the data management systems [26].

While the logical level describes the global behaviors, it is possible to allow more levels of abstraction in the implementation level. To this end, Mealy services [10] and BPEL4Chor [14] are mathematical and practical (respectively) models in the implementation level with high abstraction. Techniques for reasoning with and verification of such models have been studied in the distributed computing and verification communities [22].

A choreography is "realizable" if there is an implementation of the interacting services whose behavior is identical to the choreography. Often in service design, the choreography language for the logical level and the implementation model are already given. Ideally, each choreography can be realized and every implementation realizes a choreography, i.e., the two levels are "equivalent" in some sense.

However, the current situation is that there are many existing implementation models while new choreography languages are being developed. Instead of being the ideal case, their relationships are not clearly known. Therefore, the ability to clearly separate the logical and implementation levels hinges on understanding the fundamental relationship between choreography specifications and implementations. To this end, we phrase the following two key challenges concerning characterization of choreographies and implementations, respectively.

Challenge 1 Can we capture the set of all realizable choreographies (for a given implementation model)?

Challenge 2 Can we capture the set of implementations that realize choreographies (in a given choreography language)?

In the remainder of the paper, we give an overview of choreography models and then focus on technical problems concerning, in particular, the two challenges.

3 Choreography Models

In this section, we define the key notion of a "choreography model" and give a brief survey of several choreography modeling languages that have been studied. We divide the languages into three categories based on their underlying frameworks: finite state automata, Petri nets, and process algebras. Subsection 3.1 discusses the elements in a choreography model and related notions, and give a summary of the existing choreography modeling languages with respect to these elements. Subsections 3.2 to 3.4 provide more details of the models in each of the three categories.

3.1 Elements of a choreography model

As illustrated in Fig. 3, a choreography defines the observable interactions among the participating services. We use the words *global* and *local* to mean the behaviors or activities that are viewed in the *overall composition perspective* and in the *individual service perspective*, respectively. Thus a *choreography model* M typically has two components: a specification C of the *desired global behaviors*, and a representation I of *local services* and their local behaviors which collectively should satisfy the specified global behaviors.

A specification of desired global behaviors is called a ***choreography***, and a representation of a service is called a ***service implementation***. A *choreography modeling language* provides means to define choreography models, i.e., choreographies, service implementations, and their semantics including a mechanism to compare global behaviors generated by service implementations with a choreography. In this perspective, we informally view a choreography modeling language $\mathcal{L}$ as a collection of choreography models, $\mathcal{L} = \{(C, I) \mid C$ is a choreography and I service implementations $\}$. In the remainder of the paper, we also conveniently view $\mathcal{L}$ as a pair $\mathcal{L} = (\mathcal{C}, \mathcal{I})$, where $\mathcal{C}$ is a collection of choreographies and $\mathcal{I}$ a collection of service implementations.

A choreography can be defined using the following two types of basic elements: (1) a set of *observable actions* that happen at individual services (locally), and (2) a set of *sequencing (global) constraints* of the activities in (1).

Observable local actions are typically of two kinds: *messaging* actions for communicating with other services including sending and receiving messages, and local *activities* that are performed at individual services independent of other services. The use of activities in a choreography is primarily for organizing service operations in order to satisfy the logical requirements of a composition. For example, a "searching for books" (on a catalog) operation should happen before a "checkout" operation.

Sequencing constraints restrict the actions to specified orderings. Although these resemble the control flow constructs in programming, a notable difference is that it may not be obvious how an individual constraint on two activities in a choreography can be enforced when the activities have no connections. As we shall see in Subsection 4.2, some of such cases can be logical consequence of other constraints (that can be enforced), and others simply cannot be implemented. This is one of the interesting problems concerning choreography modeling languages.

The second component of a choreography model is a set of service implementations. When the services interact within a composition, their collective global behaviors should "conform" to the specification. In the following, we explore two essential ingredients needed in defining the notion of conformance, the latter will be given in Subsection 4.1.

Clearly, the services must communicate with each other using a *messaging model*. Different messaging models have been used in the literature. Under one model the sender of a message waits for the receiver to consume the message from the channel before it continues. The model is simple and prohibits the sender from taking any further actions, including sending another message prior to the consumption of the first by the receiver. We call this model *synchronous messaging* in the spirit of service composition. In contrast, *asynchronous messaging* models allow the sender to continue its execution

immediately after its completion of the send action. A decision to be made is how to handle the situation when new messages arrive before old messages are consumed. A natural approach is to use a *FIFO queue* for each receiver to store all its unconsumed messages in their arriving order. One can also place a limit on the size of FIFO queues, in which case, an incoming message sent to a full queue could cause one of several possible actions, including: overwrite the oldest message, delete itself, or block the sender's execution. A messaging model should clearly define the actions to be taken.

The second ingredient is how to formulate the *global behaviors of an execution of services* and compare them against a choreography. One straightforward approach is to use traces of the service executions modulo irrelevant actions. This has been studied in mostly automata based choreography models. The other is to employ the notion of *bisimulation* between the generated global behaviors and choreography. Most process algebra based choreography models adopt this approach.

Table 1 summarizes some selected choreography modeling languages in the three categories. Rows in the table indicate specific elements in the global or local specifications. Messaging and activity mean whether a choreography/service implementation can include messaging actions and non-messaging activities (respectively). Here "Global" means choreography, "Local" means service implementation, "G/L" stands for both choreography and service implementation (in all models examined, the control flow constructs are the same for choreography and service implementation). For example, in Colombo the global behavior specification could state that the "listen-to-music" activity in one service should happen before the "checkout" activity in another. The row "Messaging model" shows the model used for the languages, and the last row "Semantics" identifies whether the comparison of generated global behaviors and a choreography uses trace based semantics or bisimulation. Finally, two process algebra based modeling languages either model activities of particular types (shown as "Limited" in table) or require one service to control choice or iteration (shown as "Dominated").

Table 1. Summary of Choreography Models

	Automata			Petri nets	Process algebras		
	Mealy [10]	UML Collaboration Diagram [9]	Colombo [1]	IPN [16]	Bologna [11]	Global & Endpoint Calculi [12]	Chor & Role [25]
Global messaging	yes	yes	no	yes	yes	yes	yes
Local messaging	yes	yes	yes	yes	yes	yes	yes
Global activity	yes[1]	no	yes	no	no	yes[2]	yes
Local activity	yes[1]	no	yes	no	no	yes[2]	yes
G/L sequence	yes	yes	yes	yes	yes	yes	yes
G/L parallel	yes[1]	yes[1]	yes[1]	yes	yes	yes	yes
G/L choice	yes	no	yes	yes	yes	yes	yes[3]
G/L Recursion	yes	yes[2]	yes	yes	no	yes[3]	yes[3]
Messaging model	FIFO	FIFO	FIFO(1)	sync	sync	sync	sync
Semantics	trace	trace	trace	bisim	bisim	bisim	trace

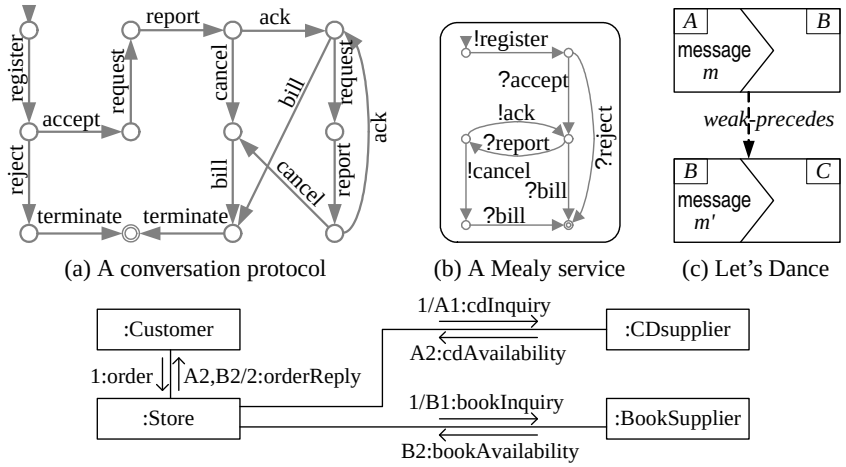
[1]Can be extended to include the element [2]Limited [3]Dominated

In the remainder of the section, we give a short survey of choreography languages based on their underlying formalisms.

3.2 Automata based models

Automata based choreography models represent both choreographies and service implementations using finite state automata (or their variants). An advantage is that state machines explicitly capture a snapshot of a composite service execution as a "state" and (local/global) behaviors can be easily captured as sequences of states in which each state transition may be associated with a message or an activity.

At the service implementation level, send and receive are modeled as message actions since they are separate individual actions while the choreography level only the status of whether a message has happened (been sent) is of interest. This group of choreography modeling languages includes conversation protocols and Mealy services [10, 19], UML collaboration diagrams [9], and the Colombo service composition model [1].



(a) A conversation protocol (b) A Mealy service (c) Let's Dance

(d) A UML collaboration diagram

Fig. 4. Automata based Choreography Modeling Languages

The use of conversations to specify choreographies was originally proposed in the IBM Conversation Support Project [20, 21]. A formal model based on this idea was developed in [10] under which a conversation protocol is represented as finite state automaton over messages, and each service as a Mealy machine over the input/output messages of the service. An example of a conversation protocol is shown in Fig. 4(a), and a Mealy service in Fig. 4(b) where the leading symbol "!" denotes an action of sending a message and "?" a receiving action. Each service has an associated FIFO queue (of unbounded capacity) for storing unconsumed incoming messages. When services are executing, a virtual global *watcher* records the sequence of messages for all send actions. A *conversation* is the sequence of messages recorded by the watcher in a successful execution. A conversation protocol is *satisfied* if every conversation by the services is a word accepted by the conversation protocol automaton.

The automata based choreography modeling approach specifies a choreography through states and transitions. It is easy to use since this approach is commonly used to specify protocols and policies. The language *Let's dance* [30, 31] provides a set of sequencing constraint primitives to allow a choreography to be specified in a graphical language. For example, Fig. 4(c) shows a message from A to B should "weak-precede" another message m', this means that B cannot send m' prior to A sending message m.

A variation of the conversation/Mealy model was studied in [9], which also uses Mealy services with unbounded FIFO queues. However, instead of conversation protocols, UML collaboration diagrams are used to specify choreographies. Fig. 4(d) illustrates a UML collaboration diagram which specifies that an "order" message is followed by "cdInquiry" and "bookInquiry" messages in any order, after their corresponding responses are made, "orderReply" can then be sent out.

Finally, another interesting variation is the Colombo model used to study an automated composition problem for semantic web services [1]. Similar to the UML model, the local services are represented as Mealy services (extended to allow OWL-S like semantic descriptions) but the message queues are limited to at most one message (size 1). Choreographies, however, are represented by finite state automata over only activities without messages. The model is an extension of the earlier "Roman" model for composing interactive web services [2].

3.3 Petri-net based models

Petri nets are another widely used tool to model, among other things, flow of control, and therefore a suitable candidate for choreography modeling languages. In [16], a Petri-net based choreography modeling language called Interaction Petri Nets (IPN) was developed. IPN treats a messaging action as a transition firing in describing a choreography. For example, Fig. 5 shows a choreography in IPN equivalent to the UML collaboration diagram in Fig. 4(d). Note that the use of Petri nets allows the concurrent Store-CDSupplier conversations and Store-BookSupplier conversations to be explicitly separated, in contrast to the UML collaboration diagram.

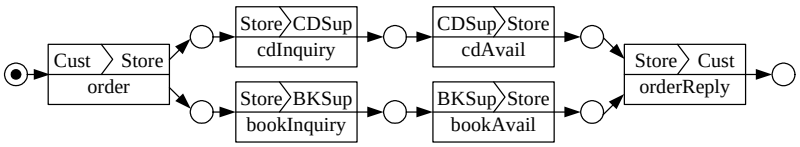


Fig. 5. Interaction Petri Nets

In [16], the technical problem studied concerns the local enforcement of an IPN choreography. In their study, the local services are represented by "behavior interfaces" that are Petri nets with "input/output" places. The services communicate with each other in the synchronous messaging model, rather than FIFO queues in automata based languages discussed in Subsection 3.2.

3.4 Process algebra based models

Recently, there have been several efforts in developing choreography modeling languages using process algebras [11, 8, 12, 25]. Common in these studies are that both choreographies and service implementations are specified in (slightly different) process algebras, with a key difference being the separation of sending and receiving a message at the local level but not at the global level, similar to the automata and Petri nets based models discussed in Subsections 3.2 and 3.3. All these approaches use the synchronous messaging model for communication.

In [11], process algebras, called the "Bologna" model in Table 1, for choreographies and service implementations (the latter are often called orchestration in process algebra based languages) were developed. The Bolonga model does not include recursion in choreography and service implementation specifications. The semantics of satisfaction of a choreography by an execution of service implementations is defined through a bisimulation between the two algebras. Global Calculus and Endpoint Calculus [12] was an attempt to provide a theoretical model for WS-CDL; they also include detailed operations and parameter passing. The semantics connecting the Global and Endpoint Calculi is also based on bisimulation. The concept of dominant role for choice and loop structures, which allows the "projection" of each choreography by inserting additional synchronization communications, was developed in Chor & Role [25]. In their model, a trace based semantics is used instead of bisimulation. Table 1 shows the basic elements available in these process algebra based languages.

4 Design and Analysis Problems

In this section, we define several key research problems concerning reasoning, design, analysis, and verification of choreographies and service implementations. In particular, we focus on the two challenges raised in Section 2 and discuss specific technical problems around the challenges.

In reasoning, we study the problem of "conformance", i.e., whether service implementations only generate global behaviors consistent with a choreography. As illustrated in Fig. 3, service design is to generate service implementations from a choreography such that the global behaviors of services are completely captured by the choreography. This "realizability" problem is important in understanding choreography specifications and thus addressing Challenge 1. A key technical problem in Challenge 2 focuses on the other direction, and demands the understanding of service implementations that realize a set of desirable choreographies. For this challenge, we formulate the "analysis" problem for service implementations.

Subsection 4.1 focuses on the conformance problem, Subsections 4.2 and 4.3 explore the problems of realizability and analysis, respectively. Subsection 4.4 outlines the main methodologies of verifying composite services.

4.1 The conformance problem

The *conformance* problem is stated as follows, assuming some fixed choreography modeling language $\mathcal{L}$: Given a choreography C in $\mathcal{L}$ and a set I of service implementa-

tions in $\mathcal{L}$, is it possible to determine if every possible execution of I always generates the behaviors allowed by C?

The conformance problem is fundamental in choreography design. It is very desirable that the problem is solvable for choreography modeling languages of interest. The problem has been studied for several choreography modeling languages.

The conformance problem is decidable for conversation protocols and Mealy services (with queues) when the queue size is bounded by some pre-determined constant, since the set of conversations of Mealy services with bounded queues is always a regular language [10]. It turned out that when the queue size restriction is removed, the problem becomes undecidable [18]; the key reason for this is that the class of finite state automata with (unbounded) FIFO queues are as computationally expressive as the class of Turing machines [7].

For process algebra based choreography languages, the problem was initially studied for the Bologna model without repetition [11]. The problem was further studied in a model extended with repetition in [8]. After projecting the choreography to local services, the checking procedure can be done locally without considering other services.

Given the known results, it appears that the conformance problem is decidable when queue sizes are bounded and undecidable for unbounded queues, in the choreography languages that have been proposed so far. In [19], it was observed that there are service compositions that need queues of size greater than 1 (possibly unbounded). It is interesting to identify classes of choreographies where the conformance problem is decidable for unbounded queues.

4.2 Realizing choreographies

The choreography approach to service design raises several new interesting questions. A key problem is whether it is possible to turn a choreography into service implementations automatically (Fig. 3). In this subsection, we discuss this "realizability" problem and other related problems.

We fix some choreography modeling language $\mathcal{L}$ and let C be a choreography. We define the following notions. The choreography C is (*weakly*) *realizable* if there exists a set I of service implementations such that the behaviors of executing I coincides with (respectively, are contained in) C. Weak realizability is useful when realizability cannot be achieved. We will discuss this notion later in the subsection.

The *realizability* problem is stated as follows: For a given choreography modeling language $\mathcal{L}$, is every choreography C in $\mathcal{L}$ (weakly) realizable? Furthermore, if C is (weakly) realizable, it is desirable to construct service implementations.

Some choreographies are not realizable. Consider the choreography $C_1 = \{m_1 m_2\}$ where service s_1 sends a message m_1 to s_3 and s_2 sends m_2 to s_3. Obviously s_2 has no way of knowing whether m_1 is sent. Thus C_1 is not realizable. Such a "missing connection" is a frequently cited reason for non-realizability (e.g., [18, 31, 12, 25, 16]). When FIFO queues are used, the reasons for non-realizability are sometimes not so obvious. Fig. 6 shows a choreography $C_2 = \{m_3 m_4 m_5, m_4 m_3\}$ over three services s_4, s_5, s_6 and messages m_3, m_4, m_5. Since every service has a FIFO queue, it was shown that every implementation that permits the two conversations in C_2 will also permit the conversation "$m_4 m_3 m_5$" that is not in C_2 [17].

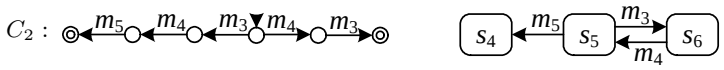


Fig. 6. An non-realizable conversation protocol

Realizability for automata-based languages was studied for conversation protocols [18, 19] and UML collaboration diagrams [9] with Mealy services and FIFO queues. For the case of conversation protocols, a sufficient condition for realizability was established in [18] which consists of three sub-conditions "lossless join", "synchronous compatible", and "autonomous". This condition was generalized to include message contents and "guarded automata" in [19]. For UML collaboration diagrams, a sufficient realizability condition is obtained in [9] which focuses on the predecessor of each *send* action. In both cases, it remains an open problem whether the realizability problem is decidable and/or a necessary and sufficient condition exists.

When queues are bounded or synchronous messaging is used (i.e., queue size is 0), the realizability problem becomes easier. For the case of conversation protocols and Mealy services, it was shown in [10] that the set of conversations of Mealy services with bounded queues is always a regular language (rather than context-sensitive for unbounded queues). This result leads to a decision procedure for realizability for bounded queues [17]. Furthermore, a necessary and sufficient condition can be formulated by modifying the sufficient condition in [18] for the unbounded queue case.

When a choreography is realizable, it is desirable to produce the service implementations. For the conversation protocols and Mealy service model, it was shown that the service implementations are simply projections of the choreography to the individual services [17]. The language Let's Dance contains a richer set of sequencing constraint primitives, projecting a choreography into local services also needs to consider the specific constraints [31]. (The referenced paper also includes a realizability checking algorithm.)

In the following, we briefly summarize recent work on realizability in process algebra and Petri nets based models, all of which assume synchronous messaging.

Realizability for process algebras was investigated primarily on the Global and Endpoint Calculi [12] and Chor & Role [25]. The main approach in these studies is to develop a "projection" operator which takes as input a choreography $\mathcal{C}$ and a participant service and produces an implementation for the service. The goal is to have the behaviors of the projected services to be identical to the choreography $\mathcal{C}$. In the context of Global and Endpoint Calculi, a sufficient condition of realizability involving connectedness, well-threadedness, and coherence was obtained [12]. In Chor & Role, a different approach was taken which consists of two parts. First, the choreography algebra uses dominated choice and repetition. Then, the projection operator inserts additional messages so that the generated services can synchronize correctly on the performed activities [25]. Intuitively, the added messages allows the dominator to communicate its decision to others, thus avoiding the missing connection problem mentioned earlier.

If a choreography C is not realizable, sometimes one could further limit the service implementations (that are obtained from projecting the choreography) so that they weakly realize C. Naturally, it is necessary to require that the generated global behaviors are not trivial (e.g., a nonempty set of traces). In fact it is also ideal that the generated behaviors should be as "close" to C as possible. In [16], the problem of weak

realizability was studied for the IPN model. The main idea is to introduce additional constraints on the service implementations so that the generated global behaviors are always allowed by the choreography. An algorithm was given for choreographies restricted to bounded Petri nets.

Before we end this subsection, it is worthwhile to mention the work of [1]. In their model, a choreography is a finite state automaton over observable (local) activities. Given a choreography C and a set of Mealy services (with "open" or configurable message channels), the choreography synthesis problem is to connect message channels among the services such that the set of observable activity sequences from executing the Mealy services is exactly the choreography C. It was shown there that the construction can be done in double exponential time complexity.

In spite of the studies on the realizability problem, there are many interesting open problems. For example, for the conversation/Mealy service model, are there restricted subclasses of Mealy services which allow unbounded queues such that the realizability problem is decidable? In general, in any choreography modeling language, if a choreography is not realizable, can we always find the "maximal" service implementations that weakly realize the choreography?

4.3 Analyzing service implementations

Subsection 4.2 discussed design problems in the top-down fashion, i.e., from choreographies to implementations. In this subsection, we focus the analysis problem, i.e., to characterize service implementations whose global behaviors are representable by choreographies.

The analysis problem may occur when one attempts to verify if given implementations e.g., a set of BPEL services, satisfy properties formulated over their global behaviors [18, 19]. If the global behaviors of the implementation can be characterized by a choreography C, reasoning and verification can be performed on C that is expected to be simpler and more efficient. In practice, it is not uncommon that one starts with a set of choreographies (a choreography language) and hopes to constrain the implementations to those that realize some of the choreographies. (We note that a related but different issue concerns verification of localized properties, e.g., concerning two services, and has been studied variously in the literature. See, e.g., the survey [27] for details.)

The *analysis* problem is defined as follows. Let $\mathcal{L} = (\mathcal{C}, \mathcal{I})$ be a choreography modeling language where $\mathcal{C}$ is a collection of choreographies and $\mathcal{I}$ a collection of implementations. Can we decide if an arbitrarily given implementation in $\mathcal{L}$ realizes some choreography in $\mathcal{C}$? Furthermore, what is the (largest) subset $\subseteq \mathcal{I}$ of implementations that realize some choreographies in $\mathcal{C}$?

Several preliminary results concerning the analysis problem have been obtained. We begin with the conversation protocols and Mealy services model of [10]. While it is known that the computation power of Mealy services is Turing complete [7], the set of conversations of a set of Mealy services is nevertheless a context sensitive language (accepted by a quasi-realtime automaton with 3 queues) [10]. Reference [10] also gives examples of Mealy services with non-regular and non-context-free sets of conversations, which are not definable by conversation protocols. It further identifies

two conditions on implementations which guarantee to produce regular sets of conversations (can be captured by conversation protocols). The first is when the queue sizes are bounded to some fixed number. In this case, each queue can be modeled as a finite state automaton and the composition can be characterized as some product machine of all Mealy services and queue automata, which turns out to be a finite state automaton. The second concerns the topology of the services and message channels. It was shown that when the graph of services and message links is a tree, the set of conversations is also a regular language.

The analysis problem was studied in [19] with $\mathcal{C}$ being the set of all conversation protocols. Motivated by the bounded queue case, the notion of "synchronizability" was formulated as follows. A set of Mealy services is *synchronizable* if its set of conversations does not change when unbounded queues are replaced with synchronous messaging (or bounded queues). A sufficient condition for synchronizability on Mealy services is identified which consists of synchronous compatible and autonomous sub-conditions.

As a final remark, the analysis problem focuses on the global behaviors of service implementations. At the first glance, this appears to simply produce the system traces. However, detailed system traces may not correspond to global behaviors permitted by choreographies, as it was shown in [10]. Secondly, having a logical representation of service behaviors is critical in many SOA applications, in fact, a key to business process design lies in the logical representations of both requirements and software processes [24].

Much of the analysis problem remains unknown. For example, is there a sufficient and necessary condition for synchronizability? Also, there are practical service implementations that are not synchronizable but they realize conversation protocols. Is it possible to characterize all conversation protocol-realizing implementations? It is also interesting to explore there problems for other choreography models.

4.4 Approaches to verification

There have been many studies on verifying compositions of web services recently (see the survey [27]). Most of these studies treat service compositions as distributed systems and properties to be verified are thus formulated over the distributed systems. Clearly the technical results from these studied are valuable contributions to the understanding of the technology that SOA can provide.

On the other hand, due to many reasons, applications of SOA in many areas including in particular business process management demand a separation of at least logical and implementation levels in process development. Verification of system properties is not sufficient if the system properties cannot be mapped to properties at the logical level. In our framework of service design, this means that we ought to be able to verify properties over choreographies of service implementations. Here we give a sampler of results along this line of verification of choreography properties.

Service design can use the top-down approach starting from a choreography. In this case, verification of logical properties can be done on the specified choreography first and if the choreography is satisfactory, it can be considered for realization [18].

We now consider the second scenario: start with service implementations (e.g., a collection of BPEL services). References [15, 28] studied the problem of checking if

a single service implementation is consistent with a choreography or other services. In [23], a choreography representing the global behaviors is first obtained which is then used for verification. This approach is not applicable for the conversation/Mealy machine model since the global behaviors are not always representable as conversation protocols [19]. Instead, analysis on service implementations is performed first and if the implementations are synchronizable, a conversation protocol can be constructed and verified.

The verification problem for services is perhaps better understood. Still, a serious challenge is the verification of services with data (from infinite domains) included. Perhaps the semantic web services approach such as OWL-S [13] of describing service semantics (with data) could help developing feasible verification approaches.

5 Conclusions

Design of web services appears to have a new twist comparing with traditional software development: having a logical level specification of global behaviors that are not identical to system traces. Such a logical-implementation level separation played a fundamental role in the development of data management techniques in the early days. It may possibly turn out to be a fundamental design principle for SOA.

In this paper we examined one aspect of the logical-implementation separation, namely how choreographies are related to the (abstract) service implementations. While prior technical results help to identify main issues on this topic, more efforts are needed to understand the different concerns at each level and to develop techniques and tools for service design.

References

1. D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. In *Proc. 31st Int. Conf. on Very Large Data Bases (VLDB)*, pages 613–624, 2005.
2. D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *Proc. 1st Int. Conf. on Service Oriented Computing (ICSOC)*, volume 2910 of *LNCS*, pages 43–58, 2003.
3. K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su. Towards formal analysis of artifact-centric business process models. In *Proceedings of 5th International Conference on Business Process Management (BPM)*, Brisbane, Australia, September 2007.
4. K. Bhattacharya, R. Guttman, K. Lymann, F. F. Heath III, S. Kumaran, P. Nandi, F. Wu, P. Athma, C. Freiberg, L. Johannsen, and A. Staudt. A model-driven approach to industrializing discovery processes in pharmaceutical research. *IBM Systems Journal*, 44(1):145–162, 2005.
5. J. Bloomberg. The seven principles of service-oriented development,. *XML & Web Services*, August 2002.
6. Business Process Execution Language for Web Services (BPEL), Version 1.1. `http://www.ibm.com/developerworks/library/ws-bpel`, May 2003.
7. D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

8. M. Bravetti and G. Zavattaro. Towards a unifying theory for choreography conformance and contract compliance. In *Proceedings of 6th International Symposium on Software Composition (SC)*, Lecture Notes in Computer Science, pages 34–50, Braga, Portugal, 2007. Springer.
9. T. Bultan and X. Fu. Specification of realizable service conversations using collaboration diagrams. In *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 122–130, Newport Beach, California, June 2007.
10. T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: A new approach to design and analysis of e-service composition. In *Proc. Int. World Wide Web Conf. (WWW)*, May 2003.
11. N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Choreography and orchestration conformance for system design. In *Proceedings of 8th International Conference on Coordination Models and Languages (COORDINATION)*, volume 4038 of *LNCS*, pages 63–81, Bologna, Italy, June 2006. Springer.
12. M. Carbone, K. Honda, N. Yoshida, R. Milner, G. Brown, and S. Ross-Talbot. A theoretical basis of communication-centred concurrent programming, 2006.
13. OWL Services Coalition. OWL-S: Semantic markup for web services, November 2003.
14. G. Decker, O. Kopp, F. Leymann, and M. Weske. BPEL4Chor: Extending BPEL for modeling choreographies. In *Proceedings of IEEE International Conference on Web Services (ICWS)*, 2007.
15. G. Decker and M. Weske. Behavioral consistency for B2B process integration. In *Proceedings of 19th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 81–95, 2007.
16. G. Decker and M. Weske. Local enforceability in interaction petri nets. In *Proceedings of 5th International Conference on Business Process Management (BPM)*, pages 305–319, 2007.
17. X. Fu. *Formal Specification and Verification of Asynchronously Communicating Web Services*. PhD thesis, University of California at Santa Barbara, 2004.
18. X. Fu, T. Bultan, and J. Su. Conversation protocols: A formalism for specification and verification of reactive electronic services. In *Proc. Int. Conf. on Implementation and Application of Automata (CIAA)*, 2003.
19. X. Fu, T. Bultan, and J. Su. Analysis of interacting BPEL web services. In *Proc. Int. World Wide Web Conf. (WWW)*, May 2004.
20. J. E. Hanson, P. Nandi, and S. Kumaran. Conversation support for business process integration. In *Proceedings of 6th IEEE Int. Enterprise Distributed Object Computing Conference*, 2002.
21. J. E. Hanson, P. Nandi, and D. W. Levine. Conversation-enabled web services for agents and e-business. In *Proceedings of the International Conference on Internet Computing (IC)*, pages 791–796, 2002.
22. R. Hull and J. Su. Tools for composite web services: A short overview. *SIGMOD Record*, 34(2):86–95, June 2005.
23. N. Lohmann, O. Kopp, F. Leymann, and W. Reisig. Analyzing BPEL4Chor: Verification and participant synthesis. In *Proceedings of International Workshop on Web Services and Formal Methods*, 2007.
24. A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.
25. Z. Qiu, X. Zhao, C. Cai, and H. Yang. Towards the theoretical foundation of choreography. In *Proceedings of 16th International World Wide Web Conference (WWW)*, pages 973–982. ACM Press, 2007.
26. R. Ramakrishnan. *Database Management Systems*. McGraw Hill, 1997.
27. F. van Breugel and M. Koshkina. Models and verification of BPEL, 2006.
28. W.M.P. van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and H.M.V. Verbeek. Conformance checking of service behavior. *ACM Transactions on Internet Technology*, 2008. (To appear).

29. Web Services Choreography Description Language Version 1.0. http://www.w3.org/TR/ws-cdl-10/, December 2004.
30. J. M. Zaha, A. Barros, M. Dumas, and A. ter Hofstede. Lets Dance: A language for service behavior modeling. In *On the Move to Meaningful Internet Systems: CoopIS, DOA, GADA, and ODBASE*, pages 145–162, 2006.
31. J. M. Zaha, M. Dumas, and A. ter Hofstede. Service interaction modeling: Bridging global and local views. In *Proceedings of IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, 2006.