E-Services: A Look Behind the Curtain

Richard Hull Michael Benedikt
Bell Labs Research
Lucent Technologies
Murray Hill, New Jersey, USA — Lisle, Illinois, USA
{hull,benedikt}@lucent.com

Vassilis Christophides
Institute for of Computer Science
Foundation for Research and
Technology-Hellas (FORTH)
Vassilika Vouton, Crete
christop@ics.forth.gr

Jianwen Su
Computer Science Department
University of California
Santa Barbara
California, USA
su@cs.ucsb.edu

ABSTRACT

The emerging paradigm of electronic services promises to bring to distributed computation and services the flexibility that the web has brought to the sharing of documents. An understanding of fundamental properties of e-service composition is required in order to take full advantage of the paradigm. This paper examines proposals and standards for e-services from the perspectives of XML, data management, workflow, and process models. Key areas for study are identified, including behavioral service signatures, verification and synthesis techniques for composite services, analysis of service data manipulation commands, and XML analysis applied to service specifications. We give a sample of the relevant results and techniques in each of these areas.

1. INTRODUCTION

The last several years have seen an explosion of activity around *electronic services*, in e-commerce, in science, and in telecommunications. The fundamental objective of e-services is clear: to have a collection of network-resident software services accessible via standardized protocols, whose functionality can be automatically discovered and integrated into applications or composed to form more complex services. Several established and emerging standards bodies are rapidly laying out the sometimes conflicting foundations that the industry will build upon.

The set of research topics behind e-services are less well-defined. What are the basic building blocks of the e-services paradigm? At a fundamental level we consider e-services as an emerging confluence of three distinct technologies: (a) process description formalisms, including automata and workflow; (b) data management (including transforms, mediation, transactions); (c) distributed computing middleware. The e-services paradigm is picking up where distributed object computing standards such as CORBA [52] left off, attempting to provide more flexible, and less structured, assembly of software components. To this end formalisms are needed to describe the visible behavior of existing e-services and to build up – perhaps automatically – new e-services that can coordinate the activities of other e-services. Data management techniques will play a more substantial role than in the past in topics (a) and (c) for the following reasons: (i) many composite e-services, especially those in e-commerce, will need to satisfy transactional properties; (ii) e-services that manipulate large data sets, especially those for data-intensive science, will need rich optimization techniques; (iii) the descriptions of e-services interfaces and compositions will be represented as (rather large) XML documents, thus paving the way for application of XML data management tools, such as high-level query languages and constraint checkers; and (iv) as auto-discovery of e-services becomes a reality we will see "queries" being made against huge volumes of intricate e-services descriptions, creating new challenges in indexing and query optimization.

It is fruitful to draw an analogy between the current evolution of e-services and the progression from structured data to semi-structured data to XML [3]. The explosion of unstructured web-resident data led from the relational model to the semi-structured model. The placement of e-services on the web has likewise led from fairly rigid technologies such as CORBA for object composition to a loosely structured framework based on, e.g., UDDI [62], SOAP [59] and WSDL [70] for e-service discovery and assembly. The lack of structure in semi-structured data raised its own problems, which were addressed by introducing mechanisms for *self-describing* data along with a minimum of additional structuring constructs; XML, DTDs and XML Schema are providing the first steps in that direction. Likewise, in e-services richer formalisms are emerging for self-description (including the pre- and post-conditions of DAML-S [31, 30], and process-based descriptions) along with more structured forms of composition (BPEL4WS [13], BPML [7], GSFL [47]).

The e-services paradigm is broad, and raises many research challenges. This paper overviews a focused subset of these research issues. Our emphasis will be on topics relevant to the database theory and verification communities, but even here the discussion will be limited in scope. These topics were chosen to underline fundamental technologies needed for the construction and analysis of composite e-services and for e-service discovery. Therefore many subjects will be omitted or covered cursorily, including transaction management, ontology-based reasoners, and planning.

Our survey will be centered around several premises about services, each of which have corollaries for theoretical research:

Services have machine-readable descriptions of their functionality, describing (among other things) their messaging behavior. In Section 2 we consider description formalisms, ranging from WSDL which gives only rudimentary information about a service's messages, to DAML-S, which offers descriptive capability so rich that it may be difficult to extract much knowledge about service behavior. We focus here on a middle-ground of *behavioral specifications* that capture the state-machine structure of a service.

New languages are required for building complex services out of *components*. In Section 3 we describe emerging "glue" formalisms for building composite e-services. These are motivated by the e-commerce, telecommunications, and scientific workflow domains. While in principle these formalisms draw on decades of research in the programming language, concurrency, and workflow communities there is nevertheless a lack of formal foundations for the emerging standards.

Having descriptions of the messaging behavior for component services permits inference of properties of composite services. In Section 4 we discuss some of the analysis issues for composite services. The feasibility of performing analysis depends on the service description formalism and the composition model. For finite-state descriptions under bounded-queue composition, standard techniques from the state machine verification community are relevant. For unbounded-queue composition and for descriptions in which message formats are not abstracted, the analysis problems are more novel.

Service descriptions can be used at service development time. A standard practice in distributed computing is to generate skeletons of implementations from signatures. In the case of e-services, the "signatures" describe messaging behavior, and it becomes natural to ask what sort of skeleton can be synthesized from a specification of (properties of) the desired global messaging behavior. As discussed in Section 5, the question has many variations depending on the service description language and the composition method.

Service descriptions are XML, and can be manipulated using XML tools. In particular, XML tools can be used in service construction, analysis, and runtime monitoring. This is discussed in Section 6.

2. DESCRIBING ATOMIC E-SERVICES

An atomic e-service consists of a set of network-accessible procedures whose functionality is described through a machine-readable description of messaging behavior. The choice of interface descriptions for services is thus critical, having consequences for service discovery, compatibility, verification, and composition. Initial standards work focused on defining the messages that could be passed in and out of the service in a single interaction, while more recent standards and research work has focused on dynamic aspects, describing or constraining the sequential behavior of the service over several interactions. Below we will give a feel of the 'space' of service description formalisms, and give a preliminary idea of the information that can be extracted from service descriptions.

2.1 Traditional input/output signatures

Our starting point for formalization of service descriptions is inspired by the W3C's Web Service Description Language (WSDL). WSDL describes an e-service via its set of visible operations. These can be thought of as message endpoints, or the set of messages that it can send and receive. WSDL specifies on the one hand "*reactive*" operations in which a message is received by the e-service. If the reactive operation is declared as "one-way" , then it does not return a response, otherwise it is a "request-response" operation, and the return type is also declared. It also describes *proactive* operations that send out messages from the e-service. "Notification" operations send out messages without waiting for a response, while "solicit-response" operations block waiting for a response, with the response type being specified with the operation. The receive and response types of the operations are mapped onto concrete XML Schema types to be used in messages. WSDL can thus be seen as an extension of traditional input/output signatures in programming languages and distributed computing to a peer-to-peer setting. An e-service is viewed both as a server (via its reactive operations) and a client (via its proactive operations).

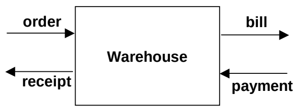


Figure 1: Warehouse with I/O signature

WSDL provides for the specification of *ports*. Ports provide the basic unit for specifying the communication linkages between e-services, i.e., it is convenient to specify that a port of one e-service is to be linked with a port of some other e-service. Ports also serve as a locus for specifying details on how web services might communicate with each other. WSDL includes a mechanism for specifying the types for error or fault values of request-notify and solicit-notify operations; in our examples we generally omit these.

EXAMPLE 2.1. *Warehouse with I/O signature.* Figure 1 shows the interface of a simple e-service for a warehouse, that can receive an *order*, send a *receipt*, send a *bill*, and receive a *payment*. As presented so far, the example remains agnostic about the service(s) that interact with the warehouse. For example, it may be that one additional service interacts with all four message classes, or it may be that a "store" service does the orders and receipts and a "bank" service receives bills and makes payments.

In a simplified syntax, these operations might be declared as follows.

```
message_class: order,
  kind: one-way,
  input: tuple [
    requester_id: string,
    invoice_id: string,
      part_list: list [
          tuple [part_id: string,
                 quantity: int,
          ]
      ]
    total_cost: dollar_amount,
    ...
    ];
message_class: receipt,
  kind: notification,
  output: tuple [ ... ];
message_class: bill,
  kind: one-way,
  input: tuple [ ... ];
message_class: payment,
  kind: notification,
  output: tuple [ ... ];
```

For this and other examples, we hide the XML encoding and other details of the syntax typically used in the standards.

The following syntax could be used to specify a warehouse service where `receipt` was a return value for `order`, and `payment` a return value for `bill`.

```
message_class: order_with_receipt,
  kind: request_response,
  input: tuple [ ... ]          // for order
  output: tuple [ ... ];        // for receipt
message_class: bill_with_payment,
  kind: solicit_response,
  output: tuple [ ... ],        // for bill
  input: tuple [ ... ];         // for payment
```

∎

The example below illustrates an e-service from data-intensive science, which typically have two classes of messages: *experimental data*, which is voluminous, and *calibration data*, which is small.

EXAMPLE 2.2. *Scientific e-service with I/O signature.* Suppose that some local authorities would like to determine the best location for installing a waste pipeline. In this scenario, the final decision (besides political reasons!) relies on the advice of environmental scientists, who need to run several simulation, statistical and visualization e-services. We assume here that one of these e-services, called Waste Transport, generates the transport characteristics of waste for a particular coastal area given a pollution source. This e-service takes as input experimental data such as local `bathymetric`, `atmospheric`, and `sea circulation` data, and produces as output `transport` data. The calibration data[1] of this e-service includes the `boundary conditions` for the particular coastal area i.e., coastline characteristics such as open ocean vs. Mediterranean basin. The Waste Transport e-service can be easily modeled using I/O signatures as presented previously, with the additional distinction between the two categories of data being used. ∎

2.2 Pre- and post-conditions

WSDL describes a single interaction with an e-service in terms of request or response messages. In service discovery, it is useful to search via properties expected of an input to the e-service or guaranteed of an output, where these properties may be based on the values, rather than the data types. For example, an e-service may require as input not just any well-formed credit card but a valid one; its output may be a boolean but it may have the effect of debiting the account associated with the credit card by $20. For this reason description formalisms like DAML-S allow the annotation of services with pre- and post-conditions expressed as propositions or external predicates of the inputs and outputs. The formalism in DAML-S is general enough that it does not require parameters in the predicates to be inputs or outputs of the service call.

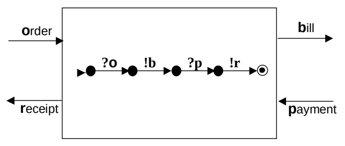
EXAMPLE 2.3. *Pre- and post-conditions for atomic e-services.* In Example 2.1 a post-condition of `Order(requester_id, invoice_id, ..., part_id)` is `ClientOf(requester_id)` $\rightarrow$ `Bill(requester_id, invoice_id, amount)`. A post-condition of `Payment(requester_id, invoice_id, amount)` is `Receipt (requester_id, invoice_id, amount)`.

The Datalog-style condition language above allows automated analysis for composite services. For example, if one also had a bank service description where `Bill(requester_id, invoice_id, amount)` has as a post-condition `account_balance (requester_id,b)` $\wedge$ `b` $\geqslant$ `amount` $\rightarrow$ `Payment(requester_id, invoice_id, amount)` then we can derive the fact that an order from a valid client with a sufficient account balance will result in a receipt. ∎

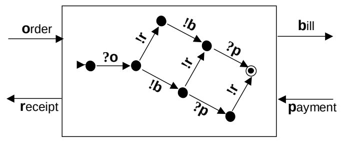
2.3 Signatures with Behavior

I/O signatures and pre-/post-conditions deal with individual service operations in isolation. However, the lifecycle of a service instance will generally consist of several related calls. Although in principal one could use DAML-S style pre- and post-conditions to fully describe sequential restrictions that occur over the course of a service, the DAML-S model provides no particular support for describing and reasoning about such sequential changes over time. This is mainly due to the lack of explicit modeling of the "states" in e-services. Such a limitation can be removed by using state machines or other reactive-system models. For example, a natural extension of the WSDL model is to describe the state changes that occur during the run of a service; that is to describe the *behavior* of the service. In full generality one can model the service as a reactive Turing machine that at any time can receive and/or output a set of messages based on its internal state, transforming its state and tape with each input or output action.

[1]In current practice the e-service must be re-compiled for each new value of the calibration data, but we expect that in future systems the parameters can be installed dynamically.



(a) "Cautious" Warehouse



(b) "Trusting" Warehouse

Figure 2: Two automata-based specifications for "warehouse"

Since the general model would prevent any kind of automation, we abstract to a more tractable formalism. The most obvious tactic is to reduce the service model to a finite state machine that can consume inputs and produce outputs. Although several concrete models have been suggested for capturing e-service behavior, from state-machine models to trace-based formalisms like Message Sequence Charts [20], the model we use here is a variant of a *Mealy machine* [42]. Our variant is a 6-tuple $\langle Q, q_0, I, O, \delta, F\rangle$, where Q is the collection of control states, with $q_0 \in Q$ the initial state and $F \subset Q$ final states, I and O are input and output alphabets both containing the distinguished element $null$, and $\delta \subset Q \times Q \times I \times O$. A Mealy machine at each state can make transitions while either consuming an input, emitting an output, both or neither, with $null$ signifying absence of either input or output. In some contexts we allow machines to have infinite executions; in other contexts we focus on executions that terminate in a final state.

EXAMPLE 2.4. *Warehouses with behavioral descriptions.* Figure 2 shows Mealy automata descriptions of two warehouse services both compliant with the signature in Example 2.1. The automaton of part (a) specifies the case where the warehouse will receive an order, then send a bill, then receive a payment, and finally send a receipt. We denote the start state of the automaton with an arrow head, and the final state(s) using a circle with small dot inside it. We follow notation typical of process algebras for indicating whether a transition is based on a receive (e.g., '`?o`') or a send (e.g., '`!b`'). Although not illustrated, "internal" transitions that don't receive or send any messages are also supported in the formal model.

In Figure 2(a) we imagine that in addition to sending a receipt, the warehouse arranges for the physical shipping of the goods to the ordering party. We describe the warehouse of part (a) as "cautious", because it does not issue a receipt until it has received payment for the order. In contrast, the automaton of Figure 2(b) allows for two activities to proceed in an interleaved fashion: the sending of a receipt (and arranging for shipment), and the sequence of sending a bill and receiving a payment. ∎

Behavioral signatures enable the application of standard techniques for automated verification to derive properties of e-services. The automata of Figure 2 model the execution of a *single* enactment of the warehouse service, i.e., a single case of receiving an order and processing it. It is also useful to consider the behavior of an e-service over time, as it performs multiple enactments. In the most general case, these enactments will be interleaved. As discussed in later sections, in some cases it is useful to model sequential execution of the enactments of a service such as warehouse. In each of the warehouse automata this can be accomplished by adding a $null$ transition from the final state to the initial state.

EXAMPLE 2.5. *Verifying properties of the Warehouse.* We give a simple example of how Mealy signatures facilitate the verification of service properties. One standard language for specifying properties is Propositional Linear Temporal Logic (LTL) [29]. LTL can describe properties of the sequences of messages (or states) that an automaton description moves through. We present some example LTL expressions[2], in the context of the warehouse automata extended with the $null$ transition from final state to initial state.

(a) G((`[?o]` $\wedge\neg$`[!b]`) X `[!b]`), i.e., "if an order has been received but a `bill` not yet sent, then in the next state a `bill` has been sent"

(b) G(`[?o]` $\rightarrow$ F(`[!b]`)), i.e., "if an order has been received then eventually a `bill` will be sent"

It is easy to see that the cautious warehouse satisfies both of these expressions, and that the trusting warehouse satisfies (b) only. ∎

The Web Services Conversation Language (WSCL) [69] provides a different approach to specifying the behavior signature of an e-service. WSCL permits specification of input/output message classes as in WSDL, and also *transitions* that are pairs of input/output messages. The intended meaning of (m_1, m_2) in a valid execution of the e-service is that a message of kind m_2 can occur immediately after a message of kind m_1. Associated with a transition there may be a condition, that refers to the kind of documents being exchanged in m_1. A WSCL conversation can be modeled as a finite state automaton whose states are the I/O message classes, and whose transitions are labeled with conditions referring to data being sent/received from the source state. It remains open to characterize the exact relationship between WSCL and Mealy automata. Note that the number of states of a WSCL automaton is bounded by the number of message classes of the service.

2.4 Working with Data

The Mealy machine model loses generality in abstracting away from the concrete message formats of WSDL into a finite collection of message classes. This subsection examines models that extend Mealy machines with information about the structure and manipulation of data.

The type theory community has considered type systems that combine the sequential behavior of message channels with traditional programming language typing of messages. Reference [41] introduces a type-checking algorithm for a rich process calculus against these "session types"; the resulting systems (see also [36]) provide incomplete methods for verifying a claimed finite-state behavior of a process.

Moving beyond just the structure of data, we can attempt to model as well the transformations being performed on the data. To maintain some tractability we will use data management oriented extensions of Mealy machines. We use a variant of the *relational machine model* of [4]. Our variant is a tuple $\langle Q, q_0, F, I, O, H, \delta\rangle$, where Q, q_0, F are as above; I, O, H are database schema; and δ is a set of elements of the form (q_1, q_2, G, T) with $q_i \in Q$. G is a boolean query over schemas I and H (the *guard*), while T is a database transformation taking as input databases conforming to I and H and producing output conforming to O and H. The idea is that a transition, if enabled by G, consumes inputs conforming to each input schema, while producing outputs conforming to each output schema and updating the hidden stores conforming to H.

The schema, query, and transformation languages used can be seen as parameters of this machine model: ideally we would want to use fragments of XML-Schema [71] and XQuery [72] for the query and transformation languages, respectively. To exhibit the kinds of analysis enabled by this model, however, we will assume relational schemas and relational calculus queries for the guard and transformation languages. We will call these *relational Mealy machines (RMMs).* This extends the model of [4] by allowing non-determinism, explicit control states, and guards.

We illustrate the notion of RMM next, using a format inspired by the relational transducers of [4]. A second illustration of RMM is given in Example 3.2.

EXAMPLE 2.6. *Warehouse with bulk data.* In this example we recast the warehouse automata of Example 2.4(a) into the framework of RMMs. Following the spirit of [4], we (i) use a variant of Datalog to express the transformation, and (ii) assume a single control state (and thus do not mention it). A sketch of the specification for the "cautious" warehouse now follows.

```
schema
  database: price, ...;
  input: order, payment;
  hidden: past_order, past_payment;
  output: bill, receipt;
state rules
  past_order(r,i,p,q,tc) +:- order(r,i,p,q,tc);
  past_payment(r,i,tc) +:- payment(r,i,tc);
output rules
  bill(r,i,tc) :- order(r,i,p,q,tc),
                  NOT past_order(r,i,p,q,tc);
                  price(p,pr), tc = mult(q,pr);
  receipt(r,i,p,q,...) :- order(r,i,p,q,tc),
                          payment(r,i,tc),
                          NOT past_payment(r,i,tc),
                          ...;
```

In this specification, the `database` is intended to model a large external database that is available for reference by the service. The `input` holds relation schemas for which input data will be received; the `hidden` holds internal data (internal states for each enactment being processed); and the `output` holds relation schemas for which output data will be produced.

Processing occurs in a sequence of phases. At each phase, a new instance for all input relations is presented. Based on this new instance, and the current values for the state relations, the `state` and `output rules` are fired in parallel to obtain new values for all of the `state` and `output` relations. We use `mult(q,pr)` to indicate the multiplication of quantity `q` by price `pr`. We have omitted some details of the rule for `receipt`, which could include information about when the items will be shipped.

The `hidden` relations are assumed to be cumulative (as suggested by the '`+:-`' operator). Intuitively, the "`past_`" relations prevent an enactment step from happening twice. For example, for each enactment (as identified by a `requester_id` and `invoice_id`) the rule for `bill` will insert a tuple into that relation during exactly one phase of the transducer execution. ∎

In comparison with Example 2.4, the RMM just described specifies the e-service in terms of *bulk* or *set-at-a-time* processing of

[2]The modal operators F, G, and X have the meaning eventually, always, next, respectively.

enactments, rather than single enactment at a time. One can think of each "move" of the RMM above as processing the data associated with multiple enactments of a Mealy e-service like that of Example 2.4. It is thus natural to analyze the more refined RMM model, that captures the efficient bulk implementation.

If a service description is published as an RMM, what analysis can be performed to validate it? This will depend on what additional information about the environment and service data is available. This supplementary information can be captured by the notion of an *instantiated relational Mealy machine (IRMM)*. An IRMM is an RMM M supplemented with a finite set $\mathcal{I}$ bounding the active domain of relations in I, and an initial database instance $\mathcal{H}$ for schema H. An IRMM as above is said to be an *IRMM based on* M. An IRMM $(M, \mathcal{I}, \mathcal{H})$ is initialized with the stores H set according to $\mathcal{H}$, and the environment restricted to send only instances with domains in $\mathcal{I}$.

Associated with an RMM M are a number of analysis problems:

- Given an IRMM *IM* based on M and LTL formula ϕ (e.g. over propositions associated with the transitions of M), does *IM* satisfy ϕ?
- Given M as above, which control states are reachable from the initial state in *some* IRMM based on M?
- Given ϕ, does there exist a relational calculus query θ over the I_j and H_j, such that θ holds on the intial state of IRMM *IM* iff *IM* satisfies ϕ?

Since every IRMM is finite state, the first problem above is decidable but with exponential complexity in the size of the initial data, assuming that the relations in I are restricted to singleton instances. The second question is undecidable for relational calculus queries by reduction to satisfiability, while the third is also undecidable in the setting of relational calculus, via reduction to the boundedness problem for Datalog. It is reasonable to look for restrictions on queries and state machine structure that render the latter two problems tractable and decrease the data complexity of the first problem. One approach would be to restrict to conjunctive queries, to fix the size of updatable internal data structures, inputs, and outputs (e.g. to have them be scalars); this is sufficient to get decidability of the second problem.

Reference [4] studies static analysis of a restricted relational machine model geared towards describing e-services. A wide range of problems are shown to be undecidable. Some decidability results are obtained for a restricted class of transducers, called "spocus"; speaking loosely these have the form illustrated in Example 2.6. The corresponding "instantiated" problems are not studied, nor the general problem of verifying properties of relational machines instantiated by data. An interesting issue is what sort of logic is appropriate for describing properties uniformly over the data. A simple fragment of First-order Linear Temporal Logic is considered in [4]. A more powerful logic would be needed to express that an e-service specified as a nondeterministic relational machine has *some* strategy for reacting to particular behaviors of the environment. Interestingly, [4] also provides a family of "log" relations, that can serve as kind of "projection" of the execution history of their restricted relational machines; results are obtained concerning whether one machine simulates another, in the sense that they produce the same logs.

3. DESCRIBING COMPOSITE E-SERVICES

The goal of this section is to describe some of the frameworks that have been proposed for combining e-services. Prior to this we develop a formal model for composition. Three key dimensions of composition are: (a) the use of bounded vs. unbounded queues, (b) the perspective of "open" vs. "closed" environments, and (c) the topology for communication between services.

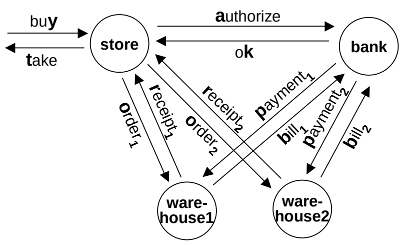


Figure 3: A composite e-service

Subsection 3.1 introduces a general *framework* for studying compositions of e-services ("e-compositions" for short), which is based on peer-to-peer communication, and considers dimensions (a) and (b). The aim of 3.1 is to motivate and develop a precise notion of composition, to define the global behavior of a composed system, and to illustrate with some examples. Subsection 3.2 then discusses approaches for "building" e-service compositions, including dimension (c). This includes discussion of approaches typical of e-commerce, scientific, and telecommunications application areas.

3.1 A Framework for Studying E-Composition

Our first example illustrates the basic components of e-services composition.

EXAMPLE 3.1. *Warehouse in a composition.* We consider a simple e-composition of four e-services, depicted in Figure 3. A (retail) Store supports interaction with customers (who are not shown); the customers `buy` goods and `take` them away. The Store also has interaction with a Bank and two warehouses, in order to replenish inventory. Figure 3 shows the four services, message classes between them, and also the two messages that can interface with customers. The two warehouses here are intended to have signatures and behaviors as in the warehouse examples of Section 2.

The typical interaction with a customer is simple: the customer issues a `buy` request and the Store responds with `take`. In a simplified scenario of inventory replenishment, the Store requests an authorization from the Bank; after receiving an approval from the Bank, the Store can send one or more orders to the Warehouses. When a Warehouse receives an order, it responds by billing the Bank for the amount on the order, and sends the Store a receipt. The Bank, in turn, makes a payment to Warehouse after receiving a bill. ∎

The example above is an *open* system, because it permits messages to and from an external *environment*. In some cases it is useful to model a composition as a *closed* system. For example, if we are interested exclusively in the interaction of the Store, the Bank, and the two Warehouses, then we might ignore the `buy` and `take` messages, and study the remainder as a closed system. This distinction, and variations on it, is important in the discussions of analysis and synthesis of e-compositions (see Sections 4 and 5.)

The intuition underlying our framework for e-compositions is shown in Figure 4. We generally use the term *peer* to denote an e-service participating in a composition. As shown there, each peer has one or more queues, that receive messages from other e-services in the composition. In terms of modeling WSDL in this framework, one could represent each WSDL port as a separate queue (although WSDL does not specify any particular implementation framework).

We now begin the formalization necessary in order define the global behavior of an e-composition. That definition will be needed

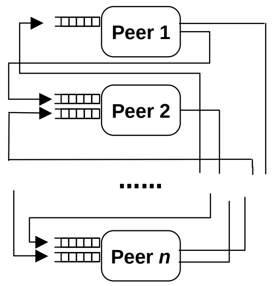


Figure 4: Abstract model of e-service composition

to discuss some of the problems of analysis and synthesis in Sections 4 and 5. Our model makes two abstractions: (a) the model essentially assumes that there is just one message queue per peer, and (b) the focus is on tracking for each message sent what *class* or *kind* of message it is (e.g., authorize, order). We focus here on closed systems. The model can be extended naturally to open systems, to distinguish between multiple queues, and to track the contents of messages.

An *e-composition schema* (*ec-schema*) is a triple (M, P, C), where M is a finite set of message classes, P a finite set of (abstract) peers (e-services), and C is a finite set of one-way communication channels. For a peer $p \in P$, a *(peer) implementation* of p is a (possibly nondeterministic) computable function which maps a sequence of incoming and outgoing messages for p into $\{\text{halt}, \text{no-op}\} \cup$ (outgoing messages from p). An *ec-implementation* of S is a mapping I such that for each $p \in P$, $I(p)$ is an implementation of p.

This definition of peer implementation is quite general; we shall consider restrictions below, e.g., that each peer implementation corresponds to a Mealy machine or an RMM, as in Section 2.

Before discussing the global behavior of peer implementations a key question is: should the queues be *bounded* or *unbounded*? Unbounded queues are studied in [15, 45, 61, 11, 12, 5, 17] and are an appropriate abstraction in cases where there might be intermittent connectivity or processing bottlenecks (e.g., because a machine has gone down). This contrasts with much previous work on communicating processes (e.g., [40, 50, 48]) in which message passing is essentially immediate or based on bounded-length queues.

Standards such as BPEL4WS do not discuss in detail which queueing models are permitted. In applications discussed in [13], threads of an e-service block while waiting for an incoming message, and the message is assumed to be processed as soon as it arrives. This suggests a model involving queues bounded to have length one. BPEL4WS also provides constructs for specifying time bounds on how long the thread will block waiting for an incoming message.

What can we take to be the global behavior of a composite service? It is typical in verification of communicating finite state machines [24] to associate with each machine a propositional labeling of states (or a labeling of transitions), and to study global behavior by examining the sequence of labels that different peers travel through. That is, the model permits a level of indirection between the states that the peers travel through and the observable behaviors associated with those states.

Another possibility, following [17], is to study the global behavior of a composition of machines by examining the language generated by messages between the peers. To this end, we posit the conceptual existence of a "watcher", which simply records each message or message class that is passed from one peer to another. So the observables are exactly all the messages. Let I be an implementation of ec-schema $S = (M, P, C)$. For the case of unbounded queues, the *ec-language* of I, denoted $\mathcal{C}(I)$, is defined to be the set of words over M that are generated by an execution of the peers in I, where each peer ends up in a final state with an empty queue. This is restricted to the case of bounded queues by looking only at words that are generated by executions in which the length of each queue never exceeds the bound.

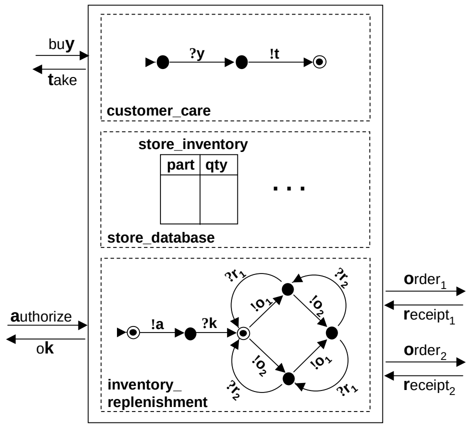


Figure 5: The store e-service

We now illustrate some of the implications of the definition of ec-language. In Example 3.3 we examine an e-composition that generates the same language under bounded and unbounded queues, and Example 3.4 shows a contrasting case.

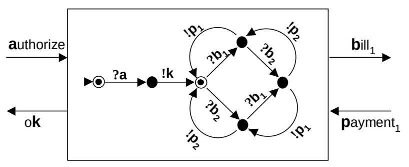
EXAMPLE 3.2. *An implementation for* Store. We consider now a peer implementation for the composition of Example 3.1. Figure 5 shows diagramatically an implementation for a simplified Store. In the Store we have two essentially independent activities, one for customer_care which is straightforward, and the other for inventory_replenishment. The latter forms part of the overall operation of the composition. An enactment of the inventory_replenishment portion of the Store first sends an `authorize` to the bank, and receives an `ok`. It may then send orders to either warehouse and receive receipts from them. Note that, according to the structure of Store, if the Store initiates an order against Warehouse$_i$, then it won't initiate a second order against Warehouse$_i$ until a receipt has been obtained for the first order.

We assume here that for the customer care and inventory replenishment automata considered separately, multiple enactments can occur in sequence. The processing of the two automata will interact via the store_database, which holds a store_inventory relation holding part numbers and quantities on hand. We assume that when customers take goods the store_inventory relation is updated, and that when quantities get below a threshold then appropriate orders are placed against one of the warehouses.

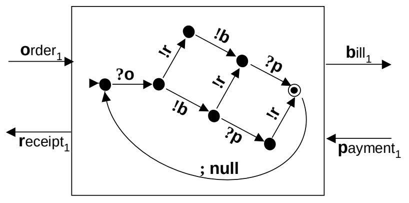
It is straightforward to formally specify the Store using an RMM. In particular, a variation on the classical cross-product construction for finite automata can be used to create the finite state control part of this RMM. ▮

We now combine the inventory replenishment portion of Store with Mealy peers for the other e-services of Figure 3.

EXAMPLE 3.3. *Language generated by a composite e-service.* Fig. 6 shows Mealy automata specifications for the Bank and Warehouse1 e-services. Warehouse2 is assumed to be analogous to Warehouse1. In one enactment of the Bank, similar to the inventory replenishment part of Store, there is a single `authorize`-`ok` pair of messages, followed by zero or more `bill`-`payment` pairs.



(a) Bank



(b) Warehouse1

Figure 6: Mealy machines for Bank **and** Warehouse1

Warehouse1 is essentially the "Trusting" warehouse of Figure 2(b), except with a $null$ move, so that arbitrarily many warehouse enactments can occur within a single "global" enactment of the composite service.

What about the language accepted by this peer implementation? It can be verified that in the case of unbounded queues, the ec-language generated by this implementation is

$$ak\ \mathrm{SH}((o_1\ \mathrm{SH}(r_1, b_1p_1))^*, (o_2\ \mathrm{SH}(r_2, b_2p_2))^*),$$

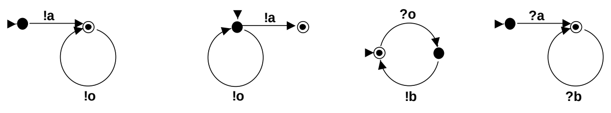
where SH is the shuffle operator. Thus, this implementation corresponds to the case where the timing of sending **r**eceipt$_1$ from Warehouse1 to Store is independent of the timing of the corresponding messages **b**ill$_1$ and **p**ayment$_1$ between Bank and Warehouse1. By using a different implementation for Warehouse1 a specific sequencing could be enforced, e.g., $o_1b_1p_1r_1$ would be enforced by use of a "cautious" Warehouse1.

What about the behavior if the queues are bounded? In this peer implementation, the sequencing of messages is tightly controlled by the interaction of the peer Mealy automata. It can be shown that in all halting executions of the composite service, all queues have length no bigger than 1, except for the queue into Bank, which may have length 2. Further, for each halting computation accepting a word w, another halting computation can be constructed that generates w and keeps the queue-length into Bank bounded by 1 (due to the structure of the automaton in Bank). So, the same language is accepted under unbounded queues and queues of length 1. ▮

In the example above, the behavior with unbounded and bounded queues is essentially the same. As illustrated now, the presence of unbounded queues can lead to different behaviors.

EXAMPLE 3.4. *Composite behavior with unbounded queues.* Consider the Mealy automata shown in Figure 7(a), (c) and (d). These can be viewed as abstracted versions of the inventory replenishment portion of Store, of Warehouse1, and of Bank of Example 3.3. However, unlike the previous case, there are no "handshake" messages between the peers. It is easily verified that if L is the ec-language generated by Store', Warehouse', and Bank' then $L \cap ao^*b^* = \{ao^nb^n \mid n \geqslant 0\}$, and so L is not regular. Also, if Store" is substituted for Store', then the language generated is precisely $\{o^nab^n \mid n \geqslant 0\}$. ▮

Results that give further insight into the structure of ec-languages are described in Sections 4 and 5.



(a) Store' (b) Store" (c) Warehouse' (d) Bank'

Figure 7: Abstract Mealy automata

3.2 Building E-Compositions

We now describe approaches for actually building e-compositions. We consider first the most general situation, called *peer-to-peer*, in which the individual e-services are essentially equals. We then move to a *mediated* approach based on a "hub-and-spoke" topology, where one service is given the special role of process *mediator*. The other services can communicate, in terms of both control and data sharing, with the mediator but not with each other. A variation is the *brokered* approach, where process control is centralized but data can pass between all of the peers. Finally, we mention some alternative topologies. A general conclusion is that in many practical cases e-compositions will have a tree-based topology, possibly with data flows that cross outside the tree structure.

Several standards and approaches have been proposed recently for specifying the composition of web services ([49, 60, 13, 7, 14, 47, 31, 22]; see also [63, 68, 66]). Although not intended to give an exhaustive description of or comparison between all of these, the discussion below reflects key aspects of the proposals and offers some contrasts between them.

We examine in turn the two main components of e-composition standards: (a) a mechanism for specifying how pairs of e-services are to be linked, and (b) a mechanism for specifying the internal process flow of an e-service.

Consider the problem of specifying a peer-to-peer e-composition as in Example 3.1. It is assumed that an e-service can be constructed to satisfy an interface specification (e.g., in WSDL), and that it can be launched without specifying the configuration details of what other running services it will interact with. Those configuration details can be given during execution, typically in the form of an XML file. The issue arises of specifying how messages from different peers are to be matched up. Should the `bill` message from the Warehouse1 be sent to the Bank or to Warehouse2? In our examples this matching is given implicitly, because of the naming conventions used. But in practice the namespaces for messages from distinct e-services will be different.

Standards such as BPEL4WS assume that the e-service message interfaces are specified using WSDL. BPEL4WS provides syntax for *service links*, which are used to specify that for the purpose of a particular e-composition, a port of one peer is linked to a port of another peer. This permits a level of indirection in the naming of ports and operations. In BPEL4WS, the XML Schema type associated with a message source may be a subtype of the type of the message target; Subsection 4.1 below discusses the issue of verifying this relationship.

While it is possible to build up an e-composition wholly from already existing services, it is common in e-commerce contexts to create a new service that coordinates the existing ones. This leads to a "hub-and-spoke" topology, where the central peer plays the role of mediator. Languages such as BPEL4WS [13] and BPML [7] are representative tools here. Figure 8 shows how the composition of Example 3.1 might be implemented under this approach, ignoring for now the interaction between the Store and customers. Note that all control and message passing flows through the mediator.

The intent of BPEL4WS, BPML, etc., is to provide a focused

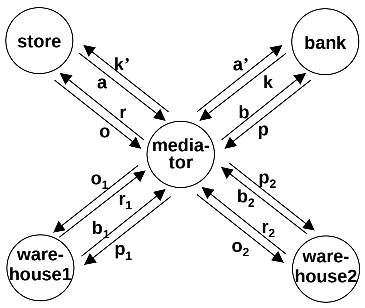


Figure 8: E-composition with hub-and-spoke topology

language for specifying the behavior of mediator e-services, rather than to provide a general-purpose programming language. Because mediators play the role of coordinating the activities of other web services there is a natural correspondence between the process flow languages for mediators and workflow languages (see [67, 65]).

The following example[3] describes how the mediator e-service of Figure 8 might be specified using a proposed standard such as BPEL4WS. Such standards provide for internal variables of type XML Schema, for the exchange of messages *à la* WSDL, supporting both send and receive, possibly with return messages, and for transfer of values between the internal variables, message inputs, and message outputs. They also provide a family of process flow constructs, such as sequencing, conditional branching, and parallelism with synchronization between threads.

EXAMPLE 3.5. *Mediated e-composition.* A simplified implementation for the Mediator of Figure 8 is shown diagrammatically in Figure 9. Part (a) of that figure shows the global structure of the Mediator. First there is processing involving the Store authorizing the Bank to make payments, followed by a do_until loop that will process orders as they arrive until some pre-established deadline arrives.

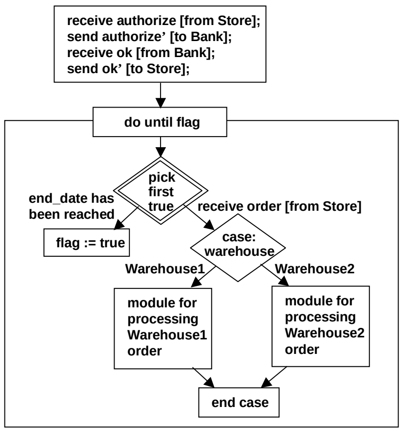
In this example the initial receive has the effect of initializing a new enactment of the overall processing of Mediator and of the e-composition in general. All other activities will occur within the context of a previously initiated enactment.

Inside the do_until loop the first step involves the "pick" construct. This allows the process to wait for the first of two or more events to take place. In this case, it waits until either a new order is received from the Store or some pre-determined end_ date has arrived. The pick construct can thus be used to ensure that a process thread is not blocked indefinitely by waiting for an incoming message that never arrives. The right-hand side of the pick is a case statement, with outcome based on whether the order is for Warehouse1 or Warehouse2. In an actual specification, in order to prevent hanging, the pick construct might be used to surround each of the receive's except for the initial one, e.g., including the receive ok from the Bank.

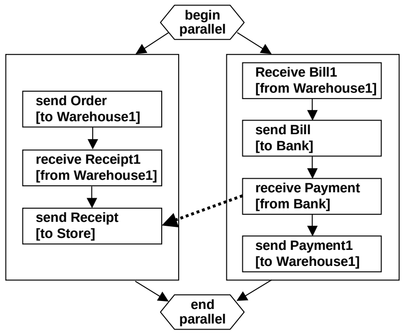
The overall effect of the do_until loop is to process orders from the Store in a sequential fashion, until the end_date has been reached.

Figure 9(b) shows the module for processing orders from Warehouse1. This illustrates the use of parallelism and synchronization between threads. The thread on the left concerns the order-processing interaction of the Mediator with Warehouse1 and the Store, and the thread on the right concerns the interaction with the Bank. The dotted arrow indicates that the send Receipt activity can begin only after the receive Payment activity has terminated.

[3]The proposed standards provide explicit mechanisms for handling faults and exceptions, along with compensations and/or rollbacks; we shall not discuss those aspects here.


receive authorize [from Store];
send authorize' [to Bank];
receive ok [from Bank];
send ok' [to Store];
do until flag
pick first true
end_date has been reached
receive order [from Store]
flag := true
case: warehouse
Warehouse1
Warehouse2
module for processing Warehouse1 order
module for processing Warehouse2 order
end case

(a) Main body


begin parallel
send Order [to Warehouse1]
receive Receipt1 [from Warehouse1]
send Receipt [to Store]
Receive Bill1 [from Warehouse1]
send Bill [to Bank]
receive Payment [from Bank]
send Payment1 [to Warehouse1]
end parallel

(b) Module for processing Warehouse1 order

Figure 9: Inside a mediator

The language generated by this e-composition is

$$aa'kk'(o\{\mathrm{SH}(o_1r_1, b_1bp)\mathrm{SH}(r,p_1), \\ \mathrm{SH}(o_2r_2, b_2bp)\mathrm{SH}(r,p_2)\})^*$$

The shuffle products here are the result of the parallel threads and synchronization links. Unlike the language of Example 3.3 there is no outer shuffle product; this follows from the form of the case statement in Figure 9(a).

In the proposed standards the synchronization relationships between parallel threads can become quite intricate, but some restrictions are imposed. For example, in BPEL4WS a synchronization link cannot cross the boundary created by a while loop. ∎

The above example illustrates how languages like BPEL4WS resemble workflow languages. In terms of expressiveness, it can be shown that a mediator built with the process flow constructs just illustrated, other than temporal conditions, can be simulated by a Mealy machine. Thus, a bounded-queue e-composition using such a mediator and Mealy peers can be simulated by a finite state automaton. The analogous result holds if the peers are RMMs. BPML, in contrast, supports a a construct to *spawn* nested enactments without necessarily waiting for them to complete. In Example 3.5 the spawn construct could be placed directly before the case construct; this would allow the Mediator to process an arbitrary number of orders in parallel. If a spawn construct is available to the mediator, then a bounded-queue composition with all other

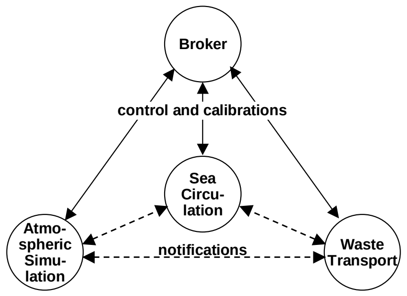


Figure 10: Brokered E-composition

peers Mealy may not be simulatable by a finite state automaton. The case for unbounded queues is considered in Section 5.

We have illustrated the mediated approach where all "leaves" are atomic e-services. We can also let some or all of the leaves be mediated e-compositions. Assuming that the new mediator is linked to exactly one e-service from each of the existing e-compositions, this will lead in most practical cases to an e-composition whose topology is a tree.

We consider now a variant of the mediated approach, called here the *brokered* approach. Here the family of peers is controlled by a single peer (the "broker"), but data can be passed between any pair of peers. This approach is embodied in GSFL [47] for scientific workflow and AZTEC [22] for telecommunications applications.

EXAMPLE 3.6. *Brokered e-composition.* Figure 10 shows the topology of a scientific workflow in which the Waste Transport service of Example 2.2 is combined with Atmospheric Simulation and Sea Circulation services. The communication between the Broker and the data-intensive services is for control and the passing of calibration data. The communication between the other services is peer-to-peer, in the form of *notifications* about data location and availability.

In this example the Sea Circulation might take as input `bathymetric` data (not shown) and data from the Atmospheric Simulation service, and the Waste Transport might take as input `bathymetric` data for the same coastal area, as well as, data from Atmospheric Simulation and Sea Circulation. The data stores, including the inputs from outside the system, are not shown. ∎

The field of scientific workflow is experiencing renewed interest due to the emergence of Grid computing [33] and the influence of the web services paradigm. Effort is currently underway to develop a coordination language [47] and system suitable for Grids. While this borrows from web services standards (e.g., by using a similar mechanism for defining message types and signatures, and for linking services together) there are three key differences: (a) the use of a broker that can direct peers to communicate with each other, (b) the emphasis on data-intensive e-services, and (c) the need to store and manipulate data provenance. We briefly discuss (c) in Section 6. With regards to (b) it is typical to compute data in batch, in part because most of the services perform aggregation operations over their input data. In most disciplines each "data product" is created by executing services along a directed acyclic graph [18]. As a result the control flow constructs needed for scientific workflow may be simpler than those used in business processes; perhaps just sequence, conditionals, and simple flavors of parallelism will suffice for the abstract model. Because many of the individual computations last days or weeks, the system underlying the abstract model must address a variety of issues involving asynchronous events and automated recovery [34]. In the Grid context where multiple computers may be capable of the same computation, there are rich opportunities for optimization. One can carefully select where and when intermediate data sets should be created.

As broker languages for scientific workflow mature it will be essential to incorporate constructs that make it easy to combine scientific workflows. What about the topology of those composite workflows? There may be a hierarchy of brokers that coordinate with each other. In addition, data-intensive services, which form the leaves of this hierarchy, may be put into pairwise notification relationships.

The AZTEC model [22] was developed for telecommunications applications, and adopts a topology based on the brokered approach. An AZTEC broker can be used to "glue" together a family of communication services (residing on, e.g., terminal devices, conference managers, media gateways) and software services (e.g., billing, web-based control interfaces). This is a brokered architecture because communication traffic and perhaps also data will pass directly between the peers being controlled by the broker.

In terms of flow of control, a chief characteristic of telecom applications is that asynchronous events from a peer may act as an interrupt for the broker and other peers. For example, many telecom services are now offered on a pre-paid basis; if the pre-pay account runs out of money then the service should be terminated as quickly as possible. Three aspects of the AZTEC model enable this: (i) instead of supporting a top-level process flow, an AZTEC specification is based on a family of flowcharts, each one in charge of handling a class of events; (ii) AZTEC allows an arbitrary number of these "event-handling" flowcharts to launch and execute simultaneously; and (iii) AZTEC provides mechanisms allowing one flowchart to pre-empt the processing of others (e.g., a "ran out of money" flowchart can abort all other flowcharts). This contrasts with "glue" languages for business e-services; in handling asynchronous events, it is typical for a BPEL4WS mediator thread to block and wait for the event. BPEL4WS supports exceptions and compensating actions which might in principle be used to support such interrupts, but to support this at least some of the interrupts would need to be viewed as exceptions at the outermost scope of the BPEL4WS specification. The exception-handling code for these would cut across the modular structure that BPEL4WS offers. Another difference between AZTEC and standards such as BPEL4WS is how synchronization between threads is achieved. In AZTEC a primary vehicle for synchronization is the use of blocking reads against a shared internal store.

The "glue" languages used in the mediated and brokered approaches are largely procedural. The emerging field of peer-to-peer databases [38, 26] supports a more declarative flavor in which query answering may involve searching across other peers to locate and obtain needed data. The messaging and evaluation strategies are not specified by the user, but are left to the execution engine. As a particular example we discuss here ActiveXML [51, 2], which provides an interesting mixture of peer-to-peer data management with XML data and the web services paradigm.

An ActiveXML document is an XML document that may include *intensional* data, that is, links to other data sources, including ActiveXML documents and web services that produce data. In response to a query against a "root" ActiveXML document, one or more of the linked ActiveXML documents might be visited. The requester may desire that all data is materialized, or that some of the data be left intensional. Reference [51] provides a framework and algorithms for providing query answers, taking into account the desired target schema and materialization policy.

What is the topology of an ActiveXML environment? At a basic level the peers can form an arbitrary graph. Since query processing is guided largely by the structure of the target (virtual, expanded) ActiveXML document, it is also useful to consider the topology from the perspective of that document. Examples and techniques in [51, 2] focus mainly on the tree-based topology associated with an ActiveXML document.

As discussed in this subsection, a variety of proposals have been made for specifying composite e-services. But each of the proposals makes use of a subset of core constructs, such as parameter passing, parallelism with restricted synchronization, and spawning of subenactments. This situation is reminiscent of the variety of semantic database models that arose in the late 1970s and early 1980s [43]. Reference [8] provides a framework to unify many of the semantic database models by providing a *meta-model* from which the models can be generated. Tools are provided so that a mapping between models M_1 and M_2 can be used to generate a mapping taking an instance and schema of M_1 into a corresponding pair for M_2. It would be worthwhile to explore whether a similar meta-model can be created to embrace the proposals for building e-compositions.

4. ANALYSIS OF E-COMPOSITIONS

A main motivation for service descriptions is that they enable automated analysis. The need for analysis is particularly acute for composite services built up from several independently-designed components: what can we conclude about a composite service based on descriptions of its components? The most basic question to ask about a set of services is: can they be composed to form a service that generates any interesting behavior at all? We call this *compatibility analysis*. A more ambitious goal is *signature generation*: obtaining a signature for a composite service. Both of these problems have variants for every service description formalism, and may also depend on the message-passing mechanism that connects services. We will discuss these analyses for several of the "generalized signature" formalisms introduced in Section 2, and for both bounded and unbounded queues. The ultimate goal is to be able to statically verify properties (e.g. in temporal logic) for composed services analogous to the ones discussed for atomic services. Verification often proceeds through signature generation, but may be solvable via more direct means.

4.1 Static Input/Output Signatures

WSDL-style signatures can be taken to imply a simple notion of compatibility in terms of *exact match* of the corresponding message schemas for receipt and sending of messages, which is trivial to check syntactically. A more liberal notion might require only *subtyping* between the incoming and outgoing message signatures. Glue languages such as BPEL4WS only require that the schema of messages for a message sender be a subtype, in the sense of language inclusion, of the schema for a message receiver. Since these languages do not necessarily mandate or expect that explicit XML Schema derivation operations are used to enforce the subtyping relationship, compatibility analysis in general may be intractable [58], thus algorithms for subtyping that are efficient in the most prevalent cases (e.g. along the lines of [54]) may be helpful.

Signature generation for the composition of compatible service is straightforward.

4.2 Behavioral Signatures

We now turn to behavioral signatures, and mention some of the basic results in three contexts: bounded-queue; unbounded-queue; and "white-box", e.g., where a mediator is described explicitly using a standard such as BPEL4WS.

For the Mealy machine model of Subsection 2.3 it is a simple matter to compute an exact signature for the bounded-queue composition of atomic services via a product construction. The number of control states will generally be exponential in the combined representation of the component machines. Compatibility analysis can also be performed. For a set of machines which exchange messages with an external environment, compatibility is probably most naturally defined as the existence of some environment for which the product machine fails to deadlock (this is analogous to the definition in [27]).

As mentioned in Section 2 verification can be performed effectively on Mealy machine signatures; for LTL this can be done in linear time in the state space of the machine. This fact combined with the above observation gives, in principle, a way to verify complex properties of concurrent systems of Mealy machines composed under bounded-queue composition. In practice this will generally be too expensive, but standard techniques from the verification community [24, 37] can be applied to avoid explicit computation of the signature in performing verification of the joint space.

We now explore the case when the message queues have no predetermined bounds. Analysis is far more difficult because finite state machines with unbounded FIFO queues are Turing complete [15], and hence nontrivial properties of such compositions are undecidable. In particular, for Mealy machines composed using unbounded queues the LTL verification problem is undecidable. In this context, signature generation must generate a Mealy machine that is only a conservative approximation to the composite signature. Existing work on abstraction techniques for unbounded queues [25] can be applied directly here. Reference [12] gives techniques for solving verification problems in this unbounded context without approximation, although they are necessarily incomplete. Developing restrictions on ec-schemas to trim the computational power of the Mealy e-compositions is possible, but the types of e-compositions will be severely limited [44].

One key aspect in an e-composition is to determine the dependencies among the messages communicated in completing an execution. For example, one can study the languages generated by an e-composition (see Subsection 3.1), e.g., asking about their position in the Chomsky hierarchy. Using an idea similar to Example 3.4, one can easily construct a Mealy ec-implementation whose ec-language is not regular nor context free. It was shown [17] that each Mealy ec-language is always context sensitive, and is accepted by some finite-state quasi-realtime automaton with 3 queues.

In the preceding discussion, the peers were viewed essentially as "black boxes". As discussed in Subsection 3.2 a common pragmatic way to build composite web services is by building a mediator or broker as a "white box". The process flow languages used are similar to typical workflow languages. Reference [64] has shown how programs from these languages can be simulated using Petri nets [53], and the same should hold for the typical e-service glue languages. The formalism of [64] corresponds to a model where peers have multiple unbounded input queues. The mapping to Petri nets permits static verification of properties such as guaranteed termination and absence of unreachable nodes. This verification can be expensive, e.g., in some models verifying node reachability can be EXPSPACE-hard.

For formalisms such as the pi-calculus, there are several typing systems [55, 41, 36] that allow one to efficiently check assertions about the behavior of a composition. The type systems range in expressiveness. Some specify only how often a particular message type can be sent on a channel. Others, such as the session-types of [41], allow the specification of sequential properties of messages. The formalisms generally require both simple and composite programs to be annotated with types, rather than allowing the type of the composition to be inferred.

4.3 Data-Intensive E-Compositions

A generalized data-processing extension of Mealy machine would have state change operations and data schemas given by XQuery and XML Schema respectively. Compatibility analysis would require both subtyping analysis of XML Schema, as for I/O signatures, and also type-checking analysis for the transformations. Since exact type-checking of even simple transformation languages is known to be undecidable [6], compatibility analysis would again have to be approximate. For the RMM model of Section 2, a simple match of relational schemas is sufficient.

For the data-enhanced RMM model, one can easily perform signature generation for the parallel composition: registers receiving input from other composed machines will become internal registers. Since static verification problems are undecidable on the full class of RMMs, the more interesting question concerns calculating signatures within restricted subclasses. Reference [4], for example, demonstrates a restricted class of "spocus transducers", for which several important static analysis problems become decidable. This class is clearly not closed under parallel composition in general, but there may be interesting subclasses that are closed.

5. SYNTHESIS OF E-COMPOSITIONS

An alternative to procedural languages like BPEL4WS for service construction is to generate implementations from declarative specification of behavior. It is not practically feasible to synthesize commercial-grade services this way, due to the difficulty in capturing behavior in sufficient detail and the computational intractability of synthesis procedures. But it is possible to generate implementation skeletons and test harnesses that can be refined into full implementations. Here we will focus the discussion on the behavioral specifications of Section 2, splitting again into the bounded and unbounded queue case.

Looking first at atomic services, the synthesis problem for finite state specifications has been studied intensely within the automata theory and verification community beginning with [16, 1]. A key distinction is between synthesizing a closed versus open system. Clearly the atomic closed case has little relevance for e-services. The synthesis problem for individual open systems where the specifications are given in LTL can be solved, but the complexity is high.

5.1 Mealy Implementations

This subsection considers first synthesis for bounded queues, for both closed and open systems. It then discuss the closed case for unbounded queues.

We start by considering synthesis of a collection of Mealy machines interacting via bounded queues. The synthesis problem here takes as input a formula and an ec-schema. As before there is a variant of the problem for open and closed systems, and in the case of e-compositions both open and closed are relevant. In the closed case, a 'folkloric' result is that synthesis from a formula can be decided by linear reduction to the satisfiability test for the logic – hence it can be done in PSPACE for LTL and in PTIME for ω-regular sets represented explicitly by an automaton. The open case is the subject of much research [57, 48]. It is shown in [57] that the problem is undecidable for general ec-schemas, but decidable for ec-schemas with linear topology. Further work [48] extends the decidability results to ec-schemas with hierarchical topology. The worst-case complexity, even for the linear case, is non-elementary.

We next address unbounded queues. When the environment is unrestricted, it is unlikely that there will be a feasible synthesis result for any interesting class of topologies. We consider here the closed case; it is hoped that the results for closed systems can be extended to open systems with constrained environments.

Let S be an ec-schema and L a language over the message classes in S. L is said to be (*strongly*) *realizable* if there exists a Mealy e-composition such that its ec-language is contained in (resp., equal to) L. The synthesis problem is to effectively construct such a Mealy e-composition from L.

Before addressing realizability we mention two properties that languages generated by (unbounded queue) e-compositions will satisfy. This provides some insight into what languages might be realizable. Recall now the first e-composition described in Example 3.4, and that Warehouse' can delay accepting the first o from its queue until Store' has produced its last o. The behavior can arise primarily because (1) a peer can make a local decision about when to pull a message off its queue. Another key fact about unbounded-queue e-compositions is that (2) a peer can make local decisions about when to output a message and hence, impact the location of that message relative to the global word being generated. In a mediated e-composition, factor (2) is reduced, but factor (1) remains and is sufficient to allow non-regular ec-languages.

To characterize Mealy ec-languages it is necessary to model the two factors just mentioned. A close examination shows that factor (2) can be dealt with by examining the "local views" by each peer of the global conversation. If Σ_p is the set of message classes that either enter or leave p, then the projection $\pi_{\Sigma_p}(w)$ of a word w over the full alphabet is obtained by removing from w all messages not in Σ_p. Given a word w, the *join* over w is the set of words $\{v \mid \text{for each peer } p, \pi_{\Sigma_p}(v) = \pi_{\Sigma_p}(w)\}$. It is easily verified that each ec-language is closed under such projection-joins. To model the effect of factor (1), a "prepone" relation between words over Σ_p can be defined. In particular, wm_2m_1w' is in the prepone of wm_1m_2w' whenever m_1 is an output from p and m_2 is an input of p. Intuitively, this corresponds to the idea that if p can produce m_1 without having m_2, then p can also produce m_1 after some other peer has put m_2 onto p's queue.

A result in [17] states that for each regular language L the closure L' of L under the projection-join and prepone operators is strongly realizable, i.e., there is a Mealy e-composition that generates L'. In general the converse is false. However, if the e-composition has a tree topology, then the converse holds [17]. Reference [35] identifies three conditions on an ω-regular language that are sufficient for the language to be strongly realizable.

We now turn to realizability. An effective characterization for this remains open. By [17] a sufficient condition for realizability of regular L is the existence of a a regular language L_1, such that the the prepone and projection-join closure of L_1 is contained in L. A syntactic characterization of these L's is not known. A special case would be to characterize the LTL formulas which define a closed language.

5.2 The Data-Intensive Context

Given the complexities involved in synthesizing even finite-state service skeletons, the idea of generating machines that perform data-manipulation commands in SQL or XQuery may seem far-fetched. However, data-integration systems ([9, 39, 21]) can be seen as performing a particular kind of data-intensive synthesis: they implement a simple declarative specification (in the form of a global view) on top of a set of "data-source" services. The interface of these components is unusual from the perspective of web-service descriptions: it is a collection of query-capabilities rather than a traditional programming language datatype, and the synthesis engine must be aware of the query language semantics.

An interesting research direction is to extend this from single global views to relational machines with queries over a global schema. The result of the synthesis will be a reactive mediator making queries to the services that provide access to source data. As in the case of traditional integration middleware, one will be interested in not simply synthesizing an arbitrary implementation (which could always done by transferring all data to the mediator), but in producing an optimal one. This will require both a notion of equivalence on RMMs and extension of cost metrics for query plans to the setting of a reactive RMM.

6. SPECIFICATIONS AS DATA

One of the main characteristics of the emerging XML dialects (e.g., WSDL, BPEL4WS, BPML, GSFL) for e-composition is the representation in terms of XML Schemas of the various process flow models employed for describing and composing e-services. A mediated e-composition is specified using a family of interrelated XML documents, including the specification of the mediator itself, the WSDL signatures of other peers, and the service links that bind them together. XML Schema types are also used to characterize the data exchanged between the external peers and the internal tasks of an e-composition. This comprehensive usage of XML provides the opportunity of using new tools to study a variety of classical problems, including type and other consistency checking, runtime analysis, and modular program construction. In turn, it is possible for the emerging standards to incorporate data elements into process flow specifications that will aid in performing consistency checking and runtime analysis.

One benefit of specifying e-compositions using XML is the possibility of verifying properties of the e-composition using XML query and constraint languages. Some representative properties that might be checked include:

(a) *Referential*. In BPEL4WS, service links should refer to peers that are participating in the e-composition.

(b) *Cardinality*. For each BPEL4WS synchronization link there should be exactly one source and one target.

(c) *Structural*. In BPEL4WS, internal synchronization links should not cross `while` scopes.

(d) *Value-based*. For every request-response operation `foo`, if a `request foo` occurs in a process then a matching `reply foo` should occur on subsequent paths.

(e) *XPath output*. Again in BPEL4WS there are requirements stating that in certain kinds of variable assignments, the XPath expression used to obtain a part of an internal variable must evaluate to a single node.

For property (a), the referential integrity constraints included in XML Schema suffice. Properties (b), (c) and (d) lie outside of the constraints under consideration by XML Schema, but they are considered in [28, 10].

Another example of (e) is checking whether the arguments of a function call will always match the signature of that function. In BPEL4WS, this involves checking whether $Q(c)$ has type T, where Q is an XPath expression, c is a container, and T is the type given by the function signature. Testing such constraints might be resolved by using a combination of XPath analysis and control flow analysis. In some cases, however, conservative static tests could be specified using (a subset of) XQuery. An interesting alternative is provided by XL [32], where all data manipulation is done by XQuery, and so one has a syntactic, conservative type checking mechanism built directly into the language.

Another kind of static analysis where XML tools may be useful involves the correlation sets found in BPEL4WS and related standards. A *correlation set* is a set of message parameters used to uniquely identify each global enactment and to prevent interference. In the case of a mediated e-composition, different peer services may use different parameter sets to identify enactments; in this case the mediator must maintain translation tables between those peers. Furthermore, in some cases a single global enactment may involve an unbounded number of enactments of some individual service or subset of services (e.g., Example 3.3, where multiple orders may arise). A challenge is to determine at compile time whether the correlation set is indeed adequate for uniquely identifying global enactments, and whether the values being assigned to the correlation set parameters will consistently and uniquely identify the global enactment of a message.

A process flow model such as BPEL4WS described using an XML Schema and a particular process flow instance – e.g., a particular BPEL4WS specification along with associated WSDL and service link specifications – can be viewed as an XML instance of that schema. But we need more than just XML Schema to capture the internal relationships that a process flow model imposes on its specifications. It would be useful to develop a family of "constraint templates" that can be associated with XML Schemas corresponding to process flow models. This might allow for the development of an automated mechanism that takes as input the augmented XML Schema of a process flow model and a specification in that model, and produces the set of constraints that must be satisfied by the XML instance. Finally, recall the discussion of meta-models for e-service glue languages from the end of Section 3. Such a meta-model should encompass the constraint templates just described.

Since an e-composition specification is an XML document, it is possible to query it using XML query languages. Going a step further, it is possible to represent an individual enactment and its state during execution using an annotated version of the specification. Reference [23] shows in the simplified context of structured workflows [46] how such runtime descriptions can be exploited to predict future behavior of the execution, such as whether a variable will be subsequently overwritten. In a structured workflow the use of control flow constructs (e.g., sequence, choice, loop, parallel) must follow a discipline of proper nesting and so there is a natural mapping of the implicit tree-like structure to XML.

XML-based representations of enactments are useful for logging their history or audit trail, with obvious applications in e-commerce. In scientific workflow the term *provenance*[4] is used to describe the history of how a "data product" was derived. In most cases the provenance of a data product can be as important as the data product itself [18]. As mentioned in Example 3.6, data products from scientific applications usually result from executing a DAG of data-intensive e-services. The DAG alone does not give the full provenance: it must be annotated with a wide variety of information, including the calibration data used for each service, information about the software used, and information about how the raw data was collected. Once a suitable model for the DAGs and annotation data is agreed upon, it should be possible to collect the provenance information into an XML data store for later querying. In practice collecting this information involves several challenges. In many scientific workflows there are human steps which must be logged. If there are rollbacks and recoveries due to system failures the provenance must reflect an accurate trace of the net effect.

So far we have discussed queries to access enactment executions and histories. What about using queries to combine e-composition specifications? Working in the context of structured workflow schemas, reference [23] makes a proposal in this direction based on

[4]We focus here on a macro form of provenance; a micro form which focuses at the level of data items is studied in, e.g., [19].

the notion of *workflow splicing*. This work supposes a library of "templates" and concrete schemas, where a template may include gaps where other templates or concrete schemas can be plugged in. XQuery can be used to construct concrete workflow schemas that satisfy given properties, such as the ability to interface with a variety of Bank. It would be fruitful to explore how far the techniques of [23] can be extended to richer e-composition formalisms, like BPEL4WS, BPML, DAML-S, and GFSL.

7. CONCLUSIONS

This paper has attempted to reveal some of the fundamental characteristics of the emerging paradigm of e-services and their composition, to identify how existing theoretical perspectives and results can be used to understand those characteristics, and to highlight directions for new research. The discussion of the paper can be summarized in three broad themes.

First, under the assumption of bounded queues, composite e-services can be studied using tools from the verification community, including automata-based models, temporal logics, and synthesis algorithms. The e-services area raises generalized versions of those problems in the context where data manipulation is incorporated, as in the Relational Mealy Machines discussed here.

Second, it is useful to study a model of composite e-services with unbounded queues. Under this assumption the behavior of e-compositions is quite different. We present some initial results in the unbounded queue case relevant to the domain of e-services, including a handful of characterizations of the global behavior of e-compositions, and some early progress towards understanding synthesis.

And third, the use of XML in many aspects of e-services raises opportunities to apply techniques from the data management community in the context of specifying compositions of e-services. These range from revisiting the classical notions of types and subtypes for input/output signatures, to the application of XQuery and XML constraint tools to e-composition specification.

This short paper was unable to address a variety of other issues raised in the new field of e-services composition. We have not considered e-service discovery beyond identifying some of the formalisms that can be used to describe e-services. We have not addressed optimization of e-compositions; a promising area is to extend optimization techniques for distributed databases to the context of scientific workflows. And we have not addressed the large and important area of transactional properties of e-compositions. We expect that in all of these areas, as with those discussed in the paper, the e-services perspective will give rise to new questions and research results in semi-structured data, integrity constraints, transaction management, optimization, and verification.

Acknowledgements

The authors thank Glenn Bruns and Patrice Godefroid for a careful reading of a draft of this paper. The work by Su is supported in part by NSF grants IIS-9817432 and IIS-0101124.

8. REFERENCES

[1] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In *Proc. of 16th Int. Colloq. on Automata, Languages and Programming*, volume 372 of *LNCS*, pages 1–17. Springer Verlag, 1989.

[2] S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu, and T. Milo. Dynamic XML documents with distribution and replication. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2003.

[3] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.

[4] S. Abiteboul, V. Vianu, B. Fordham, and Y. Yesha. Relational transducers for electronic commerce. *Journal of Computer and System Sciences*, 61(2):236–269, 2000.

[5] P.A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In *Proc. Computer Aided Verification*, pp. 305–318, 1998.

[6] N. Alon, T. Milo, F. Neven, D. Suciu, and V. Vianu. XML with data values: Typechecking revisited. In *Proc. ACM Symp. on Principles of Database Systems*, 2001.

[7] A. Arkin. *Business Process Modeling Language (BPML), Version 1.0*. BPML.org, 2002.

[8] P. Atzeni and R. Torlone. Management of multiple models in an extensible database design tool. In *Proc. Int. Conf. on Extending Database Technology*, 1996.

[9] C. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu. XML-based information mediation with MIX. In *SIGMOD*, 2000.

[10] M. Benedikt, G. Bruns, J. Gibson, R. Kuss, and A. Ng. Automated update management for XML integrity constraints. In *Proc. Workshop on Programming Languages for XML (PLAN-X)*, 2002.

[11] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In *Proc. 8th Int. Conf. on Computer Aided Verification*, volume 1102 of *LNCS*, pages 1–12. Springer-Verlag, August 1996.

[12] B. Boigelot, P. Godefroid, B. Williams, and P. Wolper. The power of QDDs. In *Proc. Fourth Static Analysis Symposium*, Sept. 1997.

[13] Business Process Execution Language for Web Services (Version 1.0). http://www.ibm.com/developerworks/library/ws-bpel, 2002.

[14] ebXML Business Process Specification Schema (Version 1.01). http://www.ebxml.org/specs/ebBPSS.pdf, May 11 2002.

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

[16] J. Buchi and L. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

[17] T. Bultan, Z. Fu, R. Hull, and J. Su. Conversation specification: A new approach to design and analysis of e-service composition. In *Proc. World Wide Web Conf.*, 2003.

[18] Workshop on data derivation and provenance, October 2002. http://www-fp.mcs.anl.gov/~foster/provenance/.

[19] P. Buneman, S. Khanna, and W.-C. Tan. Why and where: A characterization of data provenance. In *Proc. Int. Conf. on Database Theory*, 2001.

[20] CCITT. *Recommendation Z.120: Message Sequence Chart (MSC)*. CCITT, Geneva, 1992.

[21] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *Proc. of Info. Processing Society of Japan*, Tokyo, Japan, October 1994.

[22] V. Christophides, R. Hull, G. Karvounarakis, A. Kumar, G. Tong, and M. Xiong. Beyond discrete e-services: Composing session-oriented services in telecommunications. In *Proc. of Workshop on Technologies for E-Services (TES), Springer LNCS volume 2193*, Rome, Italy, September 2001.

[23] V. Christophides, R. Hull, and A. Kumar. Querying and splicing of XML workflows. In *Proc. of Intl. Conf. on Cooperating Information Systems (CoopIS)*, 2001.

[24] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[25] A. Deutsch. A storeless model for aliasing and its abstractions using finite representation of right-regular equivalence relations. In *IEEE Int. Conf. on Computer Languages*, pp. 2–13, 1992.

[26] N. Daswani, H. Garcia-Molina, and B. Yang. Open problems in data-sharing peer-to-peer systems. In *Proc. Int. Conf. on Database Theory*, pages 1–15, 2003.

[27] L. de Alfaro and T. A. Henzinger. Interface automata. In *Proc. ACM Symp. Foundations of Software Engineering*, 2001.

[28] A. Deutsch and V. Tannen. Containment for classes of Xpath expressions under integrity constraints. In *Proc. Workshop on Knowledge Representation meets Databases (KRDB)*, 2001.

[29] E. A. Emerson. Temporal and modal logic. In J. Van Leeuwen,

editor, *Handbook of Theoretical Computer Science*. Elsevier Science Publishers, 1990.

[30] DAML-S Coalition (A. Ankolekar et al). DAML-S: Web service description for the semantic web. In *The Semantic Web - ISWC 2002, Proc. 1st Int. Semantic Web Conference*, volume 2342 of *LNCS*, pages 348–363, June 2002.

[31] DAML Services Coalition (A. Ankolekar et al). DAML-S: Semantic markup for web services. In *Proc. of Int. Semantic Web Working Symposium (SWWS)*, pages 411–430, July/August 2001.

[32] D. Florescu, A. Grühagen, and D. Kossmann. XL: An XML programming language for web service specifi cation and composition. In *Proc. World Wide Web Conf.*, pages 65–76, 2002.

[33] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, 1998.

[34] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A computation management agent for multi-institutional Grids. In *Proc. IEEE Symp. High Performance Distributed Computing (HDPC)*, 2001.

[35] X. Fu, T. Bultan, and J. Su. Conversation protocols: A formalism for specifi cation and verifi cation of reactive electronic services. Submitted for publication, 2003.

[36] S. Gay and M. Hole. Types for correct communication in client-server systems. Tech. Report CSD-TR-00-07, Dept. of Computer Science, Royal Holloway, Univ. of London, Dec. 2000.

[37] P. Godefroid and D. Long. Symbolic protocol verifi cation with queue BDDs. In *Proc. IEEE Symposium on Logic In Computer Science*, pages 1–12, 1996.

[38] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What can databases do for Peer-to-Peer? In *Workshop on Databases and the Web (WebDB)*, 2001.

[39] L.M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In *Proc. Int. Conf. on Very Large Data Bases*, pages 276–285, 1997.

[40] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[41] K. Honda, V. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *Programming Languages and Systems, 7th European Symposium on Programming*, 1998.

[42] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory Languages, and Computation*. Addison-Wesley Publishing Company, Reading, Mass., 1979.

[43] R. Hull and R. King. Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys*, 19(3):201–260, September 1987.

[44] O. H. Ibarra. Reachability and safety in queue systems. In *Implementation and Application of Automata, 5th International Conference (CIAA)*, pages 145–156, July 2000.

[45] T. Jéron. Testing for unboundedness of FIFO channels. In *Proc. STACS-91: Symp. on Theoretical Aspects of Computer Science*, vol. 480 of LNCS, 322–333, Hamburg, 1991, Springer-Verlag.

[46] B. Kiepuszewski, A. ter Hofstede, and C. Bussler. On structured workfbw modelling. In *Proc. 12th Conf. on Advanced Information Systems Engineering (CAISE)*, Stockholm, Sweden, June 2000.

[47] S. Krishnan, P. Wagstrom, and G. von Laszewski. GSFL: A workfbw framework for Grid services. Technical Report Preprint ANL/MCS-P980-0802, Argonne National Laboratory, August 2002.

[48] O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *Proc. IEEE Symposium on Logic In Computer Science*, 2001.

[49] F. Leymann. Web Services Flow Language (WSFL 1.0), May 2001.

[50] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag, Berlin, 1980. Lecture Notes in Computer Science, Vol. 92.

[51] T. Milo, S. Abiteboul, B. Amann, O. Benjelloun, and F. Dang Ngoc. Exchanging intensional XML data. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2003.

[52] Object Management Group (OMG). The Common Object Request Broker: Architecture and Specifi cation (CORBA) rev. 2.6. OMG technical document see www.omg.org, 2001.

[53] J.L. Peterson. Petri nets. *ACM Comp. Surveys*, 9(3):223–251, 1977.

[54] B. Pierce and H. Hosoya. Regular expression pattern matching for XML. In *Proc. ACM Symp. on Principles of Programming Languages*, pages 67–80, 2001.

[55] B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. In *Proc. IEEE Symp. on Logic In Comp. Science*, 1993.

[56] L. Pinzon, H.-M. Hanisch, M. Jafari and T. Boucher. A comparative study of synthesis for discrete event controllers. Tech. Report, Rutgers Center for Operations Research, Rutgers U., Sept. 1997.

[57] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proc. IEEE Symp. on Foundations of Computer Science*, pages 746–757, 1990.

[58] H. Seidl. Deciding equivalence of fi nite tree automata. *SIAM Journal of Computing*, 19(3):424–437, 1990.

[59] Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ #_Toc478383487, 2000.

[60] S. Thatte. XLANG: Web services for business process design, 2001.

[61] K.J. Turner et al. *Using Formal Description Techniques – An Introduction to Estelle, Lotos and SDL*. Wiley, 1993.

[62] Universal Description, Discovery and Integration of Web Services (UDDI) 3. http://www.oasis-open.org/committees/uddi-spec/ tcspecs.shtml#uddiv3, 2002.

[63] W. M. P. van der Aalst. Don't go with the fbw: Web services composition standards exposed, *IEEE Intelligent Systems*, Jan/Feb 2003.

[64] W.M.P. van der Aalst. The application of petri nets to workfbw management. *J. of Circuits, Systems and Computers*, 8(1), 1998.

[65] W.M.P. van der Aalst, A.P. Barros, A.H.M. ter Hofstede, and B. Kiepuszewski. Advanced workfbw patterns. In *Proc. 7th Int. Conf. on Cooperative Information Systems (CoopIS 2000)*, volume 1901 of *LNCS*, pages 18–29. Springer, 2000.

[66] W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and P. Wohed. Pattern based analysis of BPML (and WSCI). Technical Report FIT-TR-2002-05, QUT, Queensland University of Technology, Brisbane, 2002. http://tmitwww.tm.tue.nl/staff/wvdaalst/Publications/p176.pdf.

[67] Workfbw management coalition. http://www.wfmc.org/.

[68] P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Pattern based analysis of BPEL4WS. Technical Report FIT-TR-2002-04, QUT, Queensland University of Technology, 2002. http://tmitwww.tm.tue.nl/staff/wvdaalst/Publications/p175.pdf.

[69] Web Services Conversation Language (WSCL) 1.0. http://www.w3.org/TR/2002/NOTE-wscl10-20020314, 2002.

[70] Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/2001/NOTE-wsdl-20010315, 2001.

[71] XML Schema, Parts 1 and 2. http://www.w3.org/TR/xmlschema-1/ and .../xmlschema-2/, May 2 2002.

[72] XQuery 1.0: An XML query language. http://www.w3.org/TR/2002/ WD-xquery-20021115/, November 15, 2002.