cl: A Language for Formally Defining Web Services Interactions

Svend Frølund, Kannan Govindarajan
HP Laboratories Palo Alto
HPL-2003-208
October 1st , 2003*

Web services have emerged recently as a distributed computing paradigm of choice for loosely-coupled computing. Current web services standards such as SOAP, and WSDL provide rudimentary mechanisms for defining interaction amongst services that may be located in different organizations. While WSDL provides the definitions for the entry-points of a service, in many cases, the interactions between services has more structure than can be described by just the definition of entry points. In particular, the sequence of interactions often is an important component of interactions between services. In current web services standards, the notion of sequencing is handled by the workflow definitions provided by proposals such as BPEL4WS. Although workflow definitions are clearly powerful enough to express all possible sequences of message exchanges between services, our approach is different. Our contention is that sharing workflow definitions across services will enable inter-operability, but leads to tighter coupling amongst the services.

We propose a conversation definition language as a simple, yet powerful, way to define web services interactions. Our definitions have no executable logic in them, just as traditional interfaces do not have any implementations in them. The main idea behind cl is to explicitly define the permissible message exchanges over time (conversations) between web services. We also introduce the notion of choice as a key enabler in expressing the externally visible behavior of services. We provide semantics for such conversation definitions in terms of the potential traces of documents exchanged between the services. We outline some of the essential properties of such conversation definitions, compare with other similar approaches, and discuss potential directions for further research.

cl: A Language for Formally Defining Web Services Interactions

Svend Frølund Kannan Govindarajan
Hewlett-Packard Laboratories, Palo Alto, CA 94304

Abstract

Web services have emerged recently as a distributed computing paradigm of choice for loosely-coupled computing. Current web services standards such as SOAP, and WSDL provide rudimentary mechanisms for defining interaction amongst services that may be located in different organizations. While WSDL provides the definitions for the entry-points of a service, in many cases, the interactions between services has more structure than can be described by just the definition of entry points. In particular, the sequence of interactions often is an important component of interactions between services. In current web services standards, the notion of sequencing is handled by the workflow definitions provided by proposals such as BPEL4WS. Although workflow definitions are clearly powerful enough to express all possible sequences of message exchanges between services, our approach is different. Our contention is that sharing workflow definitions across services will enable inter-operability, but leads to tighter coupling amongst the services.

We propose a conversation definition language as a simple, yet powerful, way to define web services interactions. Our definitions have no executable logic in them, just as traditional interfaces do not have any implementations in them. The main idea behind cl *is to explicitly define the permissible message exhanges over time (conversations) between web services. We also introduce the notion of choice as a key enabler in expressing the externally visible behavior of services. We provide semantics for such conversation definitions in terms of the potential traces of documents exchanged between the services. We outline some of the essential properties of such conversation definitions, compare with other similar approaches, and discuss potential directions for further research.*

1 Introduction

1.1 Background

Web services are a new paradigm for applications, or business processes, that span organizational boundaries and interact over the open internet through the use of standard protocols. In order for a web service in one organization to interact with a web service in another organization, one needs to establish technical conventions (i.e., standards) for interactions between web services. These technical conventions range from messaging formats (e.g., SOAP [SOA03]), interaction definitions (e.g., WSDL [WSD03]), to properties of the interactions, such as security, transactionality, etc.

Traditional models for distributed computing are typically based on synchronous communication (e.g., remote procedure calls) and shared types (e.g., interfaces specified in some interface definition language). These properties reflect the fact that traditional distributed computing models were intended for deployment within a single organization. In contrast, web services are intended to provide a distribution model for systems that span organizational boundaries

Some of the inherent characteristics of web services that differentiate them from traditional distributed computing environments are the following:

- *Loose coupling*: Changes to a web service provider should not require re-installation of software components by the clients of the web service. In effect, different players who interact should be able to evolve in a semi-independent manner. For instance, if a service provider upgrades its end, it should not require all its clients to upgrade in unison.
- *Flexible data interpretation*: The interpretation of the data communicated among enterprises is different for each enterprise. For instance, the

address field of a purchase order may have different significance for the parties. If a uniform object model is used, the semantics of data, *i.e.*, the code or logic that uses the data and hence confers semantics to the data often tends to be similar or homogeneous contributing to tighter coupling.

These characteristics often place competing requirements on the web services infrastructure. Any constraints on the semantics is likely to lead to tighter coupling, and any infrastructure that has loose coupling, and does not specify semantics adequately, can lead to loss of inter-operability or significantly increase the cost of inter-operability.

1.2 Web Service Interfaces

Just as in traditional programming systems, having a precise notion of interface is the key to achieving loose coupling *and* inter-operability in the context of web services. The emerging WSDL standard [WSD03] is the de-facto way to define web service interfaces. WSDL defines concepts, such as port, message, and operation, to define the transport, the format of the messages, and the interaction type respectively. WSDL supports a document exchange model, which means that the service interaction points are defined in terms of the documents that are exchanged with the service rather than method signature definitions. WSDL can be argued to be more extensible, and flexible than traditional interface definition mechanisms such as IDL.

As has been observed by several researchers [HBCS03, BCTH03], inter-operability between web services requires a richer notion of interface than what is offered by WSDL. In particular, it is not sufficient to specify the documents accepted by individual operations, it is also important to know the sequence in which these operations must be invoked. In other words, it is important to specify, as part of its interface, the *conversations* that a web service can engage in.

In the web-services software stack, conversations are typically described as shared workflow definitions (e.g., written in BPEL4WS [BPE03]). A workflow language is essentially a Turing complete programming language to express processes that involve multiple web services. As a light-weight alternative to workflow languages, we introduce a conversation language to describe the extensional behavior of web services. That is, our conversation language allows us to describe the sequences of interactions (i.e., conversations) that a web service may engage in, without describing the underlying logic that chooses which particular sequence to actually engage in.

In essence, our conversation language gives a declarative specification of the externally visible behavior, whereas BPEL4WS is an imperative description of the workflow associated with the service. We can thus treat a definition written in our language as a type definition for a service, and this type definition could well be implemented by a BPEL4WS workflow definition. Using a declarative, rather than an imperative, description of conversations has many advantages:

1. *Supports Ubiquitous Services*: A light-weight definition of extensional behavior enables a larger class of entities to interact. A workflow definition language that is Turing-complete places stricter requirements on the entities that interact with services.

2. *Enables Loose Coupling*: In contrast to workflow definitions, our language contains no shared logic or variables, which gives rise to a looser coupling.

3. *Improved Reasoning*: Because our language is simpler, and not Turing complete, it is simpler to reason about formally.

4. *Testing*: One of the key advantages of providing a theoretical framework is in testing the correctness of systems. The key problem in loosely-coupled systems is that the different components that make up the loosely-coupled system may be in different enterprises. This significantly increases the complexity of testing new services before they are deployed. Indeed, without a clean, minimal, externally visible interface for services, testing new services is more difficult.

Our conversation language, called cl, describes two-party conversations. The base construct in cl is the notion of an interaction: the one-way exchange of a document between two web services. A conversation is then a sequence of such interactions. A cl conversation definition is based on non-deterministic choice between, and sequential composition of, basic interactions.

One of the main challenges in defining cl is the treatment of non-deterministic choice. In languages such as CSP [Hoa78] and CCS [Mil80], processes perform a coordinated choice since they synchronize at the choice point, and agree on which branch to take. In contrast, some web service systems, such as [HBCS03], give a separate conversation specification for each web service, which means that non-deterministic choice is a purely local action. That is, the conversation definition provides no guarantee that different services will choose the same branch at a given choice point. Thus, with local choice, a conversation description has to explicitly handle the case where services make different, and possibly conflicting, choices. Dealing with potential conflicts explicitly as part of the conversation definition itself makes the conversation definition more complex. The goal of cl is to provide the semantic simplicity of coordinated choice, and at the same time provide the loose coupling afforded by local choice.

The choice semantics of cl relies on the notion of interactions that can either commit or abort. When a service reaches a choice point in its conversation definition, it performs a purely local choice between the branches. If two services choose different branches at the same choice point, one of those branches will abort and the other will commit, and the services will agree on these outcomes. The first interaction in a branch determines which service is the initial sender in that branch. An aborted interaction is only possible when different branches contain different initial senders. Moreover, the occurrence of an aborted interaction requires that the services actually choose these different branches that allow both of them to issue a send operation at about the same time. Thus, we expect aborted interactions to be rare in practice.

We organize the remainder of the paper as follows. We introduce the way we model a system of distributed web services in Section 2. We outline the syntax and semantics of our language in Section 3. We discuss the key properties of our language in Section 4, and we discuss related work in Section 5. We conclude with a discussion of future work in Section 6.

2 Model

We model a service as an entity that has two pieces: a platform and a backend application logic implementation. The platform basically executes cl definitions, performs inter-service messaging, and dispatches incoming messages to the backend application logic, etc. The backend application logic contains the application-specific behavior of a service. The boundary between the backend and the platform in a given service allows the application logic to send and receive messages through the platform, and it allows the backend to instantiate cl conversation definitions to specify the legal sequences of such send and receive actions. A given backend may instantiate multiple cl definitions, or multiple instances of the same conversation definition and thus carry out multiple conversations with different services at the same time.

A send action may either commit or abort. That is, when a backend tries to send a message as part of a conversation, the platform may either abort or commit the send. The sending of a message and signaling the outcome of the message are asynchronous actions. That is, to send a message, the backend simply deposits the message in the platform. The send then remains tentative until the platform subsequently signals that the send either commits or aborts. This asynchronous semantics for message-based communication preserves the loose coupling of services.

We model the behavior of a service in terms of the send, receive, abort, and commit events that occur between the platform and and backend of the service. We next define formally the types of events that we consider, and the types of event histories that we consider to be well-formed.

2.1 Observable Events

Since we are interested in providing a theoretical foundation of observable interactions between services, we first define the notion of observable events. These observable events are observed in the interface between the platform and the backend.

Definition 1
A *basic event* is a member of the set $\{s_P(v), r_P(v), a_P(v), c_P(v)\}$, where

$s_P(v)$ represents the event that service P sends the value v.

$r_P(v)$ represents the event that service P receives the value v.

$a_P(v)$ represents the event that service P aborts the send of value v.

$c_P(v)$ represents the event that service P commits the send of value v[1].

Note that the 'values' that are communicated between services are unconstrained. They could be XML documents, integers, etc.

Definition 2
The set of valid histories $\mathcal{H}$ is defined as follows:

1. $\lambda \in \mathcal{H}$ (λ is the history with no events in it).
2. For any basic event e, and history $h \in \mathcal{H}$, $e.h \in \mathcal{H}$.

[1]Note that the abort and commit pertain to sends and are 'status' messages pertaining to sends.

An event e is said to *occur* in a history h, if $h = h_1.e.h_2$ for some h_1 and h_2. We write $e \in h$ if event e occurs in history h. Histories with at least one event in them are said to be *non-empty.*

Definition 3
A basic event e_1 is said to *occur before* an event e_2 in a history h if there exist histories h_1, h_2, and h_3 such that $h = h_1.e_1.h_2.e_2.h_3$. We write $e_1 \prec_h e_2$ if the e_1 occurs before e_2 in h.

Definition 4
Given a basic event of the form $x_P(v)$, where $x \in \{s, r, a, c\}$, the service P is said to be the *principal actor* of the event.

Definition 5
Given two services P and Q, we define the *relevant history* of events between them to be a string of basic events where the principal actors of the events are the services P and Q.

Definition 6
A relevant history H of events between two services P and Q is said to be *locally well-formed* if the following properties hold.

1. $(r_P(v) \in H) \implies ((s_Q(v) \in H) \wedge (c_Q(v) \in H) \wedge (s_Q(v) \prec_H r_Q(v)) \wedge (s_Q(v) \prec c_Q(v)))$
2. $(a_P(v) \in H) \implies (((s_P(v) \in H) \wedge (s_P(v) \prec_H a_P(v))) \wedge (r_Q(v) \notin H))$
3. $(c_P(v) \in H) \implies ((s_P(v) \in H) \wedge (s_P(v) \prec_H c_P(v)))$

Essentially, the first component of the definition of local well-formedness states that if an event, e_1, that represents a service P receiving a document occurs in the history, an event, e_2, that represents a service Q sending the same document occurs in the history before e_1. The second component states that if an abort event occurs in a history, it must be preceded by an appropriate send event. In addition, if an abort event occurs, the relevant document is not received by the service to whom the original document was sent. The third component of well-formedness states that if a commit event occurs in a history, it is preceded by a send event. If one has guaranteed message delivery in the infrastructure, we will be able to assert that the other services will eventually receive the the committed messages as well.

From the definition of well-formedness, the following observation immediately follows:

Observation: If H is a non-empty locally well-formed history relevant to any pair of services P and Q, the first event in H is a send event.

$$\begin{aligned}
c ::=\ & P \rightarrow Q : T \mid && (1)\\
& c_1; \ldots ; c_n \mid && (2)\\
& \textit{label} \mid && (3)\\
& \textit{choice} \mid && (4)\\
& \textit{label} : \textit{choice} && (5)\\
\textit{choice} ::=\ & \textsf{either}\ c_1\ \textsf{or}\ \ldots\ \textsf{or}\ c_n\ \textsf{end} \mid && (6)\\
& \textsf{either}(P \prec Q)\ c_1\ \textsf{or}\ \ldots\ \textsf{or}\ c_n\ \textsf{end} && (7)
\end{aligned}$$

Figure 1: The abstract syntax of cl.

3 cl: a Language for Defining Conversations

3.1 Abstract Syntax

We introduce a language, called cl, to specify two-party conversations. In general, our approach can be extended to define conversations amongst an arbitrary number of parties, but we do not discuss the general case in this paper.

A conversation is an exchange of typed documents over time between two services. A conversation consists of a number of interactions. Each interaction has a sender and receiver. In cl, an interaction between the services P and Q is specified as "$P \rightarrow Q : T$," where T is the type of a document being passed from P to Q. To define such interactions, we rely on a set of services ($P, Q \in \textsf{Service}$) and a set of document types ($T \in \textsf{Type}$). We do not explicitly define the type system associated with the messages exchanged between documents, but it encompasses the type system for XML proposed in [mW03].

We define the syntax of cl in Figure 1. The figure defines the structure conversation definitions, which are elements of the set Conversation ($c \in \textsf{Conversation}$). As can been seen from the figure, there are various ways to compose conversations. Here we informally outline the meaning of composition. We define the semantics formally in Section 3.3. The conversation "$c_1; \ldots ; c_n$" is the sequential composition of the conversations $c_1 \ldots c_n$, in that order. Sequential composition means that c_{i+1} does not start until c_i has ended. The conversation "either c_1 or … or c_n end" captures branching: executing this conversation amounts to executing one of its constituent parts.

There are four kinds of branching that are possible. These are illustrated by the following examples:

either $P \rightarrow Q : T_1$ or $P \rightarrow Q : T_2$ end: This choice represents a situation where the end points of the interactions over which the choice occurs are the same, but there is a choice in the type of document that is exchanged. This choice allows service designers to model the situation when at any point in time one of a set of documents may be communicated between the services.

either $P \rightarrow Q : T_1$ or $Q \rightarrow P : T_2$ end: This choice represents a situation where the end points in the interaction are themselves different. We refer to such choice interactions as symmetric choice interactions. These interactions are critical in modeling the communication patterns that arise from timeouts, cancellations, and other interrupt-style communication patterns between services. An example is, a service orders a book from a book seller, and may want to cancel before the book-seller confirms that it can fulfill the order.

either$(P \prec Q)$ $P \rightarrow Q : T_1$ or $Q \rightarrow P : T_2$ end: This choice represents a situation where the end points of the interactions over which there is a choice are different and there is an ordering amongst the end points. This ordering dictates which end point may abort its send. We refer to such interactions as asymmetric choice interactions. The purpose is to be able to model, situations where there is asymmetry amongst interacting services. In the example from the symmetric case above, it may be the case that the even though the buyer has sent a cancel, if the cancel reaches the seller after the seller has sent the confirmation, the seller's confirmation has higher precedence. Essentially, the buyer's cancel message is rendered invalid by the seller's confirmation message.

label : either c_1; *label* or... or c_n end: This choice represents a choice with the ability to loop in one or more of the branches of the choice. This allows for potentially unbounded sequences of interactions between services. We assume a set Label of labels, and use *label* to refer to an arbitrary element of this set.

In the general case, there could be conversation fragments instead of simple interactions in the example above. In addition, the choice could be among n branches as opposed to two branches.

3.2 Example Conversation Definitions

In this subsection, we briefly define some motivating examples of conversation definitions. These example provide some of the motivation for each of the constructs that we have proposed in the language above.

3.2.1 Simple Purchase Example

In this example, we present a simplified view of the exchange of business documents in a typical business to business purchase. We focus in on the actual purchase process between a *Buyer*, and a *Seller* that are modeled as interacting web services. In this case, the type of documents exchanged are denoted by names, but can be XML schemas, simple types, etc.

```
$Buyer \rightarrow Seller : PurchaseOrder$;
$Seller \rightarrow Buyer : PurchaseOrderResponse$;
$Seller \rightarrow Buyer : AdvancedShippingNotice$;
$Seller \rightarrow Buyer : Invoice$;
$Buyer \rightarrow Seller : Payment$
```

3.2.2 Simple Purchase with Status Check

The example above illustrated the simple sequencing capability in conversation definitions. If, on the other hand, we wanted to model the possibility of the *Buyer* optionally enquiring the status of the order before sending the payment, the conversation definition can look as follows:

```
$Buyer \rightarrow Seller : PurchaseOrder$;
$Seller \rightarrow Buyer : PurchaseOrderResponse$;
$Seller \rightarrow Buyer : AdvancedShippingNotice$;
$Seller \rightarrow Buyer : Invoice$;
either
  $Buyer \rightarrow Seller : StatusEnquiry$;
  $Buyer \rightarrow Seller : Payment$
or
  $Buyer \rightarrow Seller : Payment$
end
```

3.2.3 Purchase with Cancellations Allowed

If, in addition, the *Seller* service wants to allow the buyer to to cancel orders before the shipping is done, the conversation definition may look as follows:

```
$Buyer \rightarrow Seller : PurchaseOrder$;
$Seller \rightarrow Buyer : PurchaseOrderResponse$;
either
  $Seller \rightarrow Buyer : AdvancedShippingNotice$;
  $Seller \rightarrow Buyer : Invoice$;
  either
    $Buyer \rightarrow Seller : StatusEnquiry$;
    $Buyer \rightarrow Seller : Payment$
  or
    $Buyer \rightarrow Seller : Payment$
  end
or
```

```
    $Buyer \rightarrow Seller : Cancel$
end
```

Essentially, the first choice determines whether the conversation will proceed further. If the *Cancel* branch of the choice is taken, the conversation ends, else, the conversation proceeds normally.

3.2.4 Cancellation with Asymmetric Behavior

Often, we want to allow for the situation where one of the constituent parties has greater power. For example, once a seller has shipped things, it may be significantly more expensive for the seller to cancel the shipment. In such situations, where the shipping documents from the seller is has greater *precedence* over the cancel from the buyer, the conversation definition will look as follows:

```
$Buyer \rightarrow Seller : PurchaseOrder;$
$Seller \rightarrow Buyer : PurchaseOrderResponse;$
either($Buyer \prec Seller$)
    $Seller \rightarrow Buyer : AdvancedShippingNotice;$
    $Seller \rightarrow Buyer : Invoice;$
    either
        $Buyer \rightarrow Seller : StatusEnquiry;$
        $Buyer \rightarrow Seller : Payment$
    or
        $Buyer \rightarrow Seller : Payment$
    end
or
    $Buyer \rightarrow Seller : Cancel$
end
```

3.2.5 Purchase with RFP process

Finally, before the *Buyer* sends the *Seller* the purchase order, they may have been involved in a Request for Proposal (RFP) process. The RFP process may involve any number of rounds of proposal refinement between the buyer and the seller.

```
$Buyer \rightarrow Seller : RequestForProposal;$
$Seller \rightarrow Buyer : Proposal;$
RFP : either
    $Buyer \rightarrow Seller : ProposalResponse;$
    $Seller \rightarrow Buyer : Proposal;$
    $RFP$
or
$Buyer \rightarrow Seller : ProposalAccept;$
end
$Buyer \rightarrow Seller : PurchaseOrder;$
$Seller \rightarrow Buyer : PurchaseOrderResponse;$
either($Buyer \prec Seller$)
    $Seller \rightarrow Buyer : AdvancedShippingNotice;$
    $Seller \rightarrow Buyer : Invoice;$
    either
        $Buyer \rightarrow Seller : StatusEnquiry;$
        $Buyer \rightarrow Seller : Payment$
    or
        $Buyer \rightarrow Seller : Payment$
    end
or
    $Buyer \rightarrow Seller : Cancel;$
end
```

3.3 Semantics

We define the semantics of a conversation defined in cl as the set of histories that can arise when executing the conversation between two web services. We map a conversation to a set of histories as a two-step process: we map a conversation to an intermediate representation, called a typed trace, and we then define a conformance relation between histories and typed traces. A typed trace is a sequence of *parameterized* events. That is, the elements of a typed trace contain the types of information that may be exchanged in interactions. We map a given conversation to a set of typed traces; each typed trace corresponds to a possible "unrolling" of loops and choices in the conversation. We then define a notion of conformance between histories and typed traces. Roughly speaking a history conforms to a typed trace, if the values in history events comply with the types in the typed trace, and if the sub-history at each service complies with the interaction sequencing embodied in the typed trace.

3.3.1 Typed Traces

A typed trace is a sequence of parameterized events, where each event has the following format:

Definition 7

A *parametrized event* E is a member of the set $\{S_P(V), R_P(V), A_P(V), C_P(V)\}$, where

$S_P(V)$ represents the event that service P sends a value that matches or validates against the template V.

$R_P(V)$ represents the event that service P receives a value that matches or validates against the template V.

$A_P(V)$ represents the event that service P aborts a send with a value that matches or validates against the template V.

$C_P(V)$ represents the event that service P commits a send of a value that matches or validates against the template V.

Our motivation in including the notion of values validating against a template in addition to the traditional notion of value matching a type is motivated by the recent work in type-theory for XML [mW03].

Definition 8
The set of *valid typed traces* $\mathcal{T}$ is defined as follows:

1. $\lambda \in \mathcal{T}$
2. For any parametrized event e, and trace $t \in \mathcal{T}$, $e.t \in \mathcal{T}$.

A parametrized event e is said to *occur* in a typed-trace t, if $t = t_1.e.t_2$ for some t_1 and t_2. We write $e \in t$ if event e occurs in trace t.

In essence, a typed trace is like a history, but for the fact that it has templates of documents or types of the values associated with the events as opposed to the actual values themselves. We do not define the $\prec$ relation amongst typed events in a history.

We first define semantics of conversation definitions that do not have any choice interactions in them. We later extend the semantics to include interactions that contain choice definitions in them.

Definition 9
Given an interaction i of the form $P \rightarrow Q : T$, the semantics of i is the set $\{S_P(T).C_P(T).R_Q(T)\}$. We write $i \mapsto S$, if S is the semantics of the interaction i.

In essence, given a simple interaction, there is one possible typed-trace associated with it. This typed trace captures the fact that one of the end points of the conversation did a successful send followed by a commit, while the other end of the conversation did a successful receive. The relative order of the commit and receive is not important in the typed trace.

Definition 10
Given two sets of typed traces T_1 and T_2, the *concatenation* T of T_1 and T_2 (written as $T = T_1.T_2$) is defined as follows.

$(\forall t_{1i} \in T_1)(\forall t_{2j} \in T_2)(\exists t \in T)[t = t_{1i}.t_{2j}]$
and
$(\forall t \in T)(\exists t_1 \in T_1)(\exists t_2 \in T_2)[t = t_1.t_2]$ For any set of typed traces S and integer i, $S^i = S.S.S...i$ times. For any set of typed traces S, we define $S^* = \bigcup_{i=0}^{\infty} S^i$

That is, the concatenation of two sets of typed traces is a set of traces whose elements are made up of concatenating any element of the first set with any element of the second set.

Definition 11
Given a conversation definition of the form $c = i_1; c'$ where i_1 is a simple interaction of the form $P \rightarrow Q : T$, and c' is a conversation fragment. Suppose further that $i_1 \mapsto S_{i_1}$, and $c' \mapsto S_{c'}$. The set S_c such that $c \mapsto S_c$ is defined as $S_c = S_{i_1}.S_{c'}$.

For example, consider the example conversation fragment:
$Buyer \rightarrow Seller : PO^2$;
$Seller \rightarrow Buyer : POR^3$;
$Seller \rightarrow Buyer : ASN^4$;
The semantics is:
$\{S_{Buyer}(PO).C_{Buyer}(PO).R_{Seller}(PO)\}$.
$\{S_{Seller}(POR).C_{Seller}(POR).R_{Buyer}(POR)\}$.
$\{S_{Seller}(ASN).C_{Seller}(ASN).R_{Buyer}(ASN)\}$
In essence, the typed trace corresponding to a straight-line conversation definition has exactly one element in it that is the concatenation of the typed traces of each interaction in sequence.

Now that we have the semantics of arbitratily long conversation definitions without choice, we introduce choice. We provide semantics by inducting over both the number of 'simple' interactions, and the number of choice interactions in the conversation fragment.

Suppose c is a conversation fragment that has exactly one choice interaction. In addition assume that each of the branches in the choice has at most one interaction. The possible cases are:

1. either $P \rightarrow Q : T_1$ or ... or $P \rightarrow Q : T_n$ end: In this case, suppose we can write the choice as either i_1 or ... or i_n end for interactions i_1 through i_n. Suppose further, that $\forall_{k=1}^{n} i_k \mapsto S_k$. Then either i_1 or ... or i_n end $\mapsto S$, where $S = \bigcup_{j=1}^{n} S_j$.

2. either $P \rightarrow Q : T_1$ or ... or $Q \rightarrow P : T_n$ end: First consider the case when $n = 2$. That is, the choice is of the form either $P \rightarrow Q : T_1$ or $Q \rightarrow P : T_2$ end. Now, suppose we designate $i_1 = P \rightarrow Q : T_1$, and $i_2 = Q \rightarrow P : T_2$. We know that $i_1 \mapsto S_1 = \{S_P(T_1).C_P(T_1).R_Q(T_1)\}$, and $i_2 \mapsto S_2 = \{S_Q(T_2).C_Q(T_2).R_P(T_2)\}$. We define the set S that either $P \rightarrow Q : T_1$ or $Q \rightarrow P : T_2$ end maps to as follows: $S = S_L \cup S_R$, where S_L is the set

   $S_1 \cup S_3$ Where S_3 is: $\{S_P(T_1).S_Q(T_2).C_P(T_1).A_Q(T_2).R_Q(T_1)\}$

   and S_R is the set:

   $S_2 \cup S_4$ Where S_4 is the set: $\{S_P(T_1).S_Q(T_2).C_Q(T_2).A_P(T_1).R_P(T_2)\}$

   Definition 12
   Two interactions i, and i', are said to *interfere*

[2] We use PO to represent PurchaseOrder in our examples in this section.

[3] We use POR to represent PurchaseOrderResponse in our examples in this section.

[4] We use ASN to represent AdvancedShippingNotice in our examples in this section.

if the sender of one interaction is the receiver in the other interaction. In the example above, the interactions i_1, and i_2 interfere. The sets $S3$, and S_4, are said to be the *interference sets* for the interactions i_1 and i_2. The set S_3 is referred to as the i_1-*biased interference set*, whereas S_4 is referred to as the i_2-*biased interference set*. The interference set for two interactions that do not interfere is the empty set ϕ.

Now, suppose that there are two branches in the choice c , but each branch has a number of interactions. Suppose one branch is of the form $i_1; c_1$, and another branch is of the form $i_2; c_2$. Suppose $i_1 = P \rightarrow Q : T_1$ and $i_2 = Q \rightarrow P : T_2$. Suppose that either $P \rightarrow Q : T_1$ or $Q \rightarrow P : T_2$ end $\mapsto S_L \cup S_R$ Suppose in addition that $c_1 \mapsto S_{c_1}$, $c_2 \mapsto S_{c_2}$ (c_1 and c_2 are simple linear conversation fragments). The semantics S of c is: $S_L.S_{c_1} \cup S_R.S_{c_2}$.

If the choice has n branches, k of which are conversation fragments whose first interaction of the form $P \rightarrow Q : T_k, k = 1..l$, and m are fragments whose first interaction of the form $Q \rightarrow P : T_j, j = 1..m$. That is the interaction is of the form: either c_1 or $\ldots$ c_k or c_{k+1} or $\ldots$ c_{k+m} end where $k + m = n$. Suppose the first interactions in each branch are $i_1, \ldots, i_k, i_{k+1}, \ldots, i_{k+m}$. Suppose further that the rest of the conversation in each branch is: $c'_1, \ldots, c'_k, c'_{k+1}, \ldots, c'_{k+m}$. Suppose further that for each interaction $i_x, x = 1..n, i_x \mapsto S_{i_x}$ and for each fragment $c'_y, y = 1..n, c'_y \mapsto S_{c'_y}$.

Suppose further that for any interaction i_k, the set S_{kz} represents the i_k-biased interference set between interactions i_k, and i_z.

The semantics of the choice is $\bigcup_{a=1}^{n}(S_{i_a} \cup \bigcup_{b=1}^{n} S_{ab}).S_{c'_a}$. Note that this definition is powerful enough to provide semantics for both type choice and symmertric choice interactions.

For example, consider the following conversation fragment that has a simple type choice interaction:

either
  $Buyer \rightarrow Seller : SE$[5];
  $Buyer \rightarrow Seller : Pay$[6]
or
  $Buyer \rightarrow Seller : Pay$
end

The semantics is: $(\{S_{Buyer}(SE).C_{Buyer}(SE).R_{Seller}(SE)\}.$ $\{S_{Buyer}(Pay).C_{Buyer}(Pay).R_{Seller}(Pay)\})$ $\cup\{S_{Buyer}(Pay).C_{Buyer}(Pay).R_{Seller}(Pay)\}$.

Now, consider a fragment, that has a symmetric choice:

either
  $Seller \rightarrow Buyer : ASN$;
  $Seller \rightarrow Buyer : Invoice$;
or
  $Buyer \rightarrow Seller : Cancel$
end

The semantics associated with the fragment is: $(\{S_{Seller}(ASN).C_{Seller}(ASN).R_{Buyer}(ASN)\}.$ $\{S_{Seller}(Invoice).C_{Seller}(Invoice).R_{Buyer}(Invoice)\})$ $\cup\{S_{Buyer}(Cancel).C_{Buyer}(Cancel).R_{Sellerer}(Cancel)\}$ $\cup\{S_{Seller}(ASN).S_{Buyer}(Cancel).C_{Seller}(ASN).$ $A_{Buyer}(Cancel).R_{Buyer}(ASN)\}$ $\cup\{S_{Seller}(ASN).S_{Buyer}(Cancel).C_{Buyer}(Cancel).$ $A_{Seller}(ASN).R_{Seller}(ASN)\}$

3. either($P \prec Q$) $P \rightarrow Q : T_1$ or $\ldots$;or $Q \rightarrow P : T_n$ end: First consider the case when $n = 2$. That is, the choice is of the form either($P \prec Q$) $P \rightarrow Q : T_1$ or $Q \rightarrow P : T_2$ end. Now, suppose we designate $i_1 = P \rightarrow Q : T_1$, and $i_2 = Q \rightarrow P : T_2$. We know that $i_1 \mapsto S_1 = \{S_P(T_1).C_P(T_1).R_Q(T_1)\}$, and $i_2 \mapsto S_2 = \{S_Q(T_2).C_Q(T_2).R_P(T_2)\}$. We define the set S that either($P \prec Q$) $P \rightarrow Q : T_1$ or $Q \rightarrow P : T_2$ end maps to as follows: $S = S_L \cup S_R$, where S_L is the set S_1, and S_R is the set: $S_2 \cup S_4$ Where S_4 is the set: $\{S_P(T_1).S_Q(T_2).C_Q(T_2).A_P(T_1).R_P(T_2)\}$

   The set S_4, the i_2-biased interference set, is referred to as the *asymmetric interference set* for $Q \rightarrow P : T_2$ in either($P \prec Q$) $P \rightarrow Q : T_1$ or $Q \rightarrow P : T_2$ end. In an asymmetric interaction of the form either($P \prec Q$) $P \rightarrow Q : T_1$ or $Q \rightarrow P : T_2$ end, the i_1-biased interference set is defined to be the empty set.

   Now, suppose that there are an two branches in the choice c , but each branch has a number of interactions. Suppose one branch is of the form $i_1; c_1$, and another branch is of the form $i_2; c_2$. Suppose $i_1 = P \rightarrow Q : T_1$ and $i_2 = Q \rightarrow P : T_2$. Suppose that either($P \prec Q$) $P \rightarrow Q : T_1$ or $Q \rightarrow P : T_2$ end $\mapsto S_L \cup S_R$ Suppose in addition that $c_1 \mapsto S_{c_1}$, $c_2 \mapsto S_{c_2}$ (c_1 and c_2 are simple linear conversation fragments). The semantics S of c is: $S_L.S_{c_1} \cup S_R.S_{c_2}$, where S_L, and S_R are defined as above.

[5]We use SE to represent StatusEnquiry in our examples in tis section.

[6]We use Pay to represent Payment in our examples in tis section.

If the choice has n branches, k of which are conversation fragments whose first interaction of the form $P \to Q : T_k, k = 1..l$, and m are fragments whose first interaction of the form $Q \to P : T_j, j = 1..m$. That is the interaction is of the form: either c_1 or ... c_k or c_{k+1} or ... c_{k+m} end where $k + m = n$. Suppose the first interactions in each branch are $i_1, \ldots, i_k, i_{k+1}, \ldots, i_{k+m}$. Suppose further that the rest of the conversation in each branch is: $c'_1, \ldots, c'_k, c'_{k+1}, \ldots, c'_{k+m}$. Suppose further that for each interaction $i_x, x = 1..n, i_x \mapsto S_{i_x}$ and for each fragment $c'_y, y = 1..n, c'_y \mapsto S_{c'_y}$.

In addition, suppose that for interactions $i_l, l = k+1 \ldots n$, let $S_{i_l i_o}, o = 1 \ldots k$ represent the asymmetric interference set between interation i_l, and i_o. Using this, we can define the complete asymmetric interference set S_l^{asym} for interaction i_l as follows $S_{i_l}^{asym} = \bigcup_{r=1}^{k} S_{i_l i_r}$.

The semantics of the choice is $\bigcup_{a=1}^{k}(S_{i_a}).S_{c'_a} \bigcup \bigcup_{b=1}^{m}(S_{i_{k+b}} \cup S_{i_{k+b}^{asym}}).S_{c'k+b}$.

For example, consider a conversation fragment of the form:

either($Buyer \prec Seller$)
  $Seller \to Buyer : ASN$;
  $Seller \to Buyer : Invoice$;
or
  $Buyer \to Seller : Cancel$
end

The semantics associated with the fragment is:
$(\{S_{Seller}(ASN).C_{Seller}(ASN).R_{Buyer}(ASN)\}.$
$\{S_{Seller}(Invoice).C_{Seller}(Invoice).R_{Buyer}(Invoice)\})$
$\cup\{S_{Buyer}(Cancel).C_{Buyer}(Cancel).R_{Seller}(Cancel)\}$
$\cup\{S_{Seller}(ASN).S_{Buyer}(Cancel).C_{Seller}(ASN).$
$A_{Buyer}(Cancel).R_{Buyer}(ASN)\}$

Essentially, if the *Buyer* has lower precedence than the *Seller*, the *Buyer* has to be prepared to abort its send.

4. *label* : either c_1; *label* or... or c_n end: If the loop choice has n branches, k of which are conversation fragments that loop and m are fragments that dont loop. That is the interaction is of the form: either c_1 or ... c_k or c_{k+1} or ... c_{k+m} end where $k + m = n$. Suppose the first interactions in each branch are $i_1, \ldots, i_k, i_{k+1}, \ldots, i_{k+m}$. Suppose further that the rest of the conversation in each branch is: $c'_1, \ldots, c'_k, c'_{k+1}, \ldots, c'_{k+m}$. Suppose further that for each interaction $i_x, x = 1..n, i_x \mapsto S_{i_x}$ and for each fragment $c'_y, y = 1..n, c'_y \mapsto S_{c'_y}$.

Suppose further that for any interaction i_k, the set S_{kz} represents the i_k-biased interference set between interactions i_k, and i_z.

The semantics of the loop choice can be written as:
$(\bigcup_{a=1}^{k}(S_{i_a} \cup \bigcup_{b=1}^{n} S_{ab}))^*.(\bigcup_{d=k+1}^{n}(S_{i_d} \cup \bigcup_{e=1}^{n} S_{de}))$

For example, consider a conversation fragment of the form:

RFP : either
  $Buyer \to Seller : PR$[7];
  $Seller \to Buyer : Pl$[8];
  RFP
or
  $Buyer \to Seller : PA$[9];
end

The semantics associated with the fragment is:
$(\{S_{Buyer}(PR).C_{Buyer}(PR).R_{Seller}(PR)\}.$
$\{S_{Seller}(P1).C_{Seller}(Pl).R_{Buyer}(Pl)\})^*.$
$S_{Buyer}(PA).C_{Buyer}(PA).R_{Seller}(PA)$

Now that we have the machinery in place for providing the semantics of conversation fragments with at most one choice statement, we can easily generalize it to a conversation fragment that has any number of choice statements in a straightforward inductive manner. We rely on the following properties of conversation definitions.

Lemma 1
Given a conversation fragment f_1 of the form:

either
  either c_{i1} or ... or c_{il} end
or
  c_1
or
  ⋮
or
  c_{k-1}
end

(where each c_x is a conversation fragment) and a conversation fragment f_2 of the form: either c_{i1} or ... c_{il} or c_1 or ... c_{k-1} end, for any typed trace t, $f_1 \mapsto t \iff f_2 \mapsto t$.

Proof Sketch
The proof relies on the observation that for each possible typed trace in the semantics of the nested frag-

[7]We use PR to denote ProposalResponse in the examples in this section.
[8]We use Pl to denote Proposal in the examples in this section.
[9]We use PA to denote ProposalAccept in the examples in this section.

ment, there is a typed trace in the semantics of the flattened fragment, and vice versa.

Lemma 2
Given a conversation fragment f_1 of the form $\mathsf{either}\ c_1\ \mathsf{or}\ \ldots\ c_k\ \mathsf{or}\ c_{k+1}\ \mathsf{or}\ \ldots\ c_n\ \mathsf{end}$, where each of the branches $c_1, \ldots, c_n$ have the same first interaction, there is an equivalent definition of the fragment where the first interaction in each branch in the choice does not have the same first interaction.

Proof Sketch
Essentially, if each of the branches has the same first interaction, say, i_1. Therefore, for all $i, i = 1..n, c_i = i_1; c_i'$. Consider the fragment f_2 of the form $i_1; \mathsf{either}\ c_1'\ \mathsf{or}\ \ldots\ c_k'\ \mathsf{or}\ c_{k+1}'\ \mathsf{or}\ \ldots\ c_n'\ \mathsf{end}$. For any typed trace t, $f_1 \mapsto t \iff f_2 \mapsto t$.

Theorem 1
If the first interaction of one of the branches of a choice interaction is itself a choice interaction, there exists an equivalent conversation fragment where the first interaction none of the branches of a choice interaction is itself not a choice interaction.

Proof Sketch
From the lemma above, we have have the base case. We can induct on the number of such nested choice interactions to get the general result.

3.3.2 History Conformance

We define what it means for a locally well-formed history to conform to a typed trace. The conformance relation has two aspects: there is a structural aspect, which requires that the sequence of interactions agree between a history and a typed trace, and there is a value aspect, which requires that the data exchanged as part of the history validates against the templates of the typed trace.

To define history conformance, we first define two notions of projection. For any locally well-formed history h, we define the projection of h onto a service P, as the subhistory of h where all events have P as principal actor. That is, in constructing the projection, we simply drop all events that do not have P as principal actor. We write the projection of h onto P as $h|P$. If no events in h have P as principal actor, the projection is the empty history. Similarly, we define the projection of a typed trace t onto a service P as the subtrace of t where all parameterized events have P as principal actor. We use $t|P$ to denote this projection.

In Figure 2, we define a conformance relation between histories and traces. Given a history h and a typed trace t, if "$h \to t$" we say that h conforms to t. Essentially, a history conforms to a typed trace if the sub-history that happens at each service conforms to the sub-trace defined for that service. By focussing on

$$\lambda \to \lambda \tag{8}$$

$$\frac{\forall P \in \mathsf{Service} : h|P \to t|P}{h \to t} \tag{9}$$

$$\frac{v : V \quad v' : V' \quad h \to t}{s_P(v).c_P(v').h \to S_P(V).C_P(V').t} \tag{10}$$

$$\frac{v : V \quad v' : V' \quad h \to t}{s_P(v).a_P(v').h \to S_P(V).A_P(V').t} \tag{11}$$

$$\frac{v : V \quad h \to t}{r_P(v).h \to R_P(V).t} \tag{12}$$

Figure 2: Defintion of $\to$

"local" conformance, we avoid defining conformance relative to an external observer. In particular, the conformance relation does not have to account for the incidental interleaving of events at distributed services. We use the notation "$v : V$" to indicate that the value v validates against the template V.

We can extend the notion of conformance to conversation definitions. We say that a history h conforms to a conversation definition c if h conforms to one of c's typed traces:

$$h \to c \Leftrightarrow \exists t : (c \mapsto S \wedge t \in S) \tag{13}$$

4 Key Results

In this section, we formally outline some of the key theorems and outline proofs for the same.

Theorem 2
Given a conversation definition c, there exists a unique set of histories H such that $h \in H \iff h \to c$.

Proof Sketch
This is a consequence of how the $\to$ relation is defined.

The main consequence of the theorem above is that the semantics of conversation definitions in terms of histories is unique and well-defined.

Theorem 3
There exists a decision procedure p that given a history h and a conversation definition c can decide whether $h \to c$.

Proof Sketch
This can be proven by a straight forward induction on the length of h or by constructing the decision procedure.

This property is more specific to conversation definitions. If one were to represent the protocol between two services as a workflow with shared variables, and a turing-complete language for expressing computation, as is the case with BPEL4WS [BPE03], such properties would be infeasible to prove.

Theorem 4
Given a conversation definition c and a history h such that $\neg(h \rightarrow c)$, there is a deterministic procedure to determine the first event $e \in h$ and interaction i in c such that the mismatch between i and c is the first instance that causes $\neg(h \rightarrow c)$.

Proof Sketch
This property essentially ensures that there is a way to determine the cause of deviation of a history from a conversation definition.

The main consequence of this theorem is that it is possible to have a procedure to *debug* conversation definitions. Proving a similar procedure for generic workflow definitions may not be feasible.

Essentially, the last two theorems tell us that it is possible to decide whether a trace conforms to a conversation definition. In addition, if a history does not conform to a conversation definition, we can determine the first cause of non-conformance. These properties therefore provide a basis of managing, and testing web services interactions.

5 Related Work

There are many web servies related standards that are emerging that are relevant for this work. We enumerate some of the main efforts and briefly describe the relationship with our work.

- *SOAP and WSDL*: SOAP and WSDL [SOA03, WSD03], are the standards that are emerging out of the web services community as standards for enabling web services interactions. There are a couple of key differences between our work and SOAP/WSDL. cl is a language for defining the externally visible behavior of services and is complementary in many respects to SOAP, and WSDL. cl also supports a wider variety of basic interaction types by virtue of supporting choice interactions. The symmetric and asymmetric choice interactions supported by cl have no analogue in WSDL. We believe that these choice interactions play a key role in enabling loose coupling while allowing for interoperability.

- *WSCL and SFS*: The work on WSCL [WSC02] served as the basis for the language cl. The authors were among the key contributors to WSCL, as well as a more complete stack of specifications that comprised the Service Framework Specification [SFS01]. Our work provides some of the formal underpinnings for WSCL. In addition, we consider more general kinds of choice interactions than were allowed in WSCL.

- *BPEL4WS*: BPEL4WS [BPE03] has emerged as a proposal from leading web services infrastructure providers as a proposal for describing business process execution. Version 1.1 of the specification [BPE03] has added support for ability to describe business protocols as a special case of business processes. The key difference between our approaches is that cl does not have any executable logic embedded in it unlike BPEL4WS. This distinction is a result of cl representing the externally visible behavior and BPEL4WS being a richer language in which the protocols can also be expressed. The simplicity of cl makes it easier to reason about.

- *Other Research*: Richard Hull, et al [HBCS03, BFHS03] have proposed a mealy-machine-based formalism to express the behavior of web services. Similar to cl, a mealy machine can express the behavior of a web services as sequences of interactions, where the sequences are constructed by sequential composition and non-deterministic choice. Unlike mealy machines, cl provides programming constructs to deal with situations where web services make conflicting choices. If the mealy machines in two web services take conflicting paths, the conflict resolution must be described explicitly in the subsequent interactions. In contrast, cl provides a notion of aborted sends and asymmetric choice to more conveniently handle conflicting choices. In addition, the mealy-machine formalism uses *null* transitions to model the internal state transitions of the services. Our conversation definitions in cl are not meant to model the internal transitions of the service. This separation of concerns, we believe provides the basis for inter-operability amongst services while preserving loose coupling. The separation between the internal and external behavior of a service also explains the relationship between cl and the body of work in modeling concurrent and mobile systems [Mil99].

6 Conclusions and Directions for Further Research

In this paper, we have outlined a simple language cl for describing the externally visible behavior of web services, provided semantics for the same. The semantics of a conversation was the set of well-formed histories of observable events. We also discussed the relationship of cl to other emerging web services technologies.

There are many possible directions for further research.

- *Multiple Services:* We believe our approach of defining interactions amongst web services is applicable to interactions among any number of services. For instance, our current definitions of interactions allow for one sender and one receiver. We can extend the language definition to accomodate multiple senders or receivers with appropriate barrier/synchronization or broadcast semantics. Since we are interested in extensional behavior, and are not interested in modeling the internal states of the services, we believe that our approach will be different from traditional approaches to this problem.

- *Reliability:* A core construct to provide reliability within an enterprise is the notion of a transaction. A transaction is a programming abstraction that allows consistent updates of shared data, such as bank accounts or inventory data, in the presence of failures and concurrency. A transaction typically satisfies the ACID properties: it is atomic with respect to failures, it transforms one consistent state to another, it executes in logical isolation from other transactions, and its effects remain durable. In the context of web services, the consistency, isolation, and durability are often the responsibility of the backend. Atomicity, on the other hand, does have an impact on the external behavior of the service, and we can provide such support in cl.

- *Implementation:* We believe that implementing cl as part of a web services infrastructure is also an area for further investigation. An implementation would provide further evidence for the thesis that a language like cl will provide a means for achieving interoperability in the presence of loose coupling.

- *Type Systems:* In this paper, we have laid some of the foundations for why conversation definitions can be viewed as interface defitions for services that are interacting. Our definition of the semantics of these conversation definitions may provide a basis of a type-theory for interacting services. We may be able define notions such as sub-typing, extensions, etc., providing a formal type-system for services that goes beyond the traditional notion of interface as type definition.

References

[BCTH03] B. Benatallah, F. Casati, F. Toumani, and R. Hamadi. Conceptual modeling of web service conversations. In *Proceedings of CAiSE 2003*, 2003.

[BFHS03] Tevfik Bultan, Xiang Fu, Richard Hull, and Jianwen Su. Conversation specification: A new approach to design and analysis of e-service composition. In *Proceedings of the World Wide Web Conference*, 2003.

[BPE03] *Business Process Execution Language for Web Services version 1.1.* http://www-106.ibm.com/developerworks/library/ws-bpel/, 2003.

[HBCS03] Richard Hull, Michael Benedikt, Vassilis Christophides, and Jianwen Su. E-services: A look behind the curtain. In *Proceedings of ACM Symposium on Principles of Database Systems.* ACM, 2003.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, August 1978.

[Mil80] R. Milner. *A Calculus of Communicating Systems.* Springer Verlag, 1980. LNCS 92.

[Mil99] R. Milner. *Communicating and Mobile Systems: the π-calculus.* Cambridge University Press, 1999.

[mW03] Jèrôme Simèon and Philip Wadler. The Essence of XML. In *Proceedings of 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages.* ACM, 2003.

[SFS01] *Service Framework Specification.* http://www.hpl.hp.com/techreports/2001/HPL-2001-138.html, 2001.

[SOA03] *SOAP Version 1.2, Part 0: Primer.* http://www.w3.org/TR/soap12-part0/, 2003.

[WSC02] *Web Services Conversation Language (WSCL) 1.0.* http://www.w3.org/TR/2002/NOTE-wscl10-20020314/, 2002.

[WSD03] *Web Services Description Language (WSDL) Version 1.2 Part 1: Core Language.* http://www.w3.org/TR/2003/WD-wsdl12-20030611/, 2003.