A formal account of contracts for Web services

S. Carpineti[1], G. Castagna[2], C. Laneve[1], and L. Padovani[3]

[1] Department of Computer Science, University of Bologna
[2] École Normale Supérieure de Paris
[3] Information Science and Technology Institute, University of Urbino

Abstract. We define a formal contract language along with subcontract and compliance relations. We then extrapolate contracts out of processes, that are a recursion-free fragment of CCS. We finally demonstrate that a client completes its interactions with a service provided the corresponding contracts comply. Our contract language may be used as a foundation of Web services technologies, such as WSDL and WSCL.

1 Introduction

The recent trend in Web services is fostering a computing scenario where loosely coupled parties interact in a distributed and dynamic environment. Such interactions are typically sequences of messages that are exchanged between the parties. The environment, being dynamic, makes it not feasible to define or assemble parties statically. In this context, it is fundamental for clients to be able to search at run-time services with the required capabilities, namely the format of the exchanged messages, and the protocol – or *contract* – required to interact successfully with the service. In turn, services are required to publish such capabilities in some known repository.

The Web Service Description Language (WSDL) [6, 5, 4] provides a standardized technology for describing the interface exposed by a service. Such description includes the service location, the format (or *schema*) of the exchanged messages, the transfer mechanism to be used (i.e. SOAP-RPC, or others), and the contract. In WSDL, contracts are basically limited to one-way (asynchronous) and request/response (synchronous) interactions. The Web Service Conversation Language (WSCL) [1] extends WSDL contracts by allowing the description of arbitrary, possibly cyclic sequences of exchanged messages between communicating parties.

Both WSDL and WSCL documents can be published in repositories [2, 7] so that they can be searched and queried. However, this immediately poses an issue related to the *compatibility* between different published contracts. It is necessary to define precise notions of contract similarity and compatibility and use them to perform service discovery in the same way as, say, type isomorphisms are used to perform library searches [18, 8]. Unfortunately, neither WSDL nor WSCL can effectively define these notions, for the very simple reason that they do not provide any formal characterization of their contract languages. This cries out for a mathematical foundation of contracts and the formal relationship between clients and contracts.

In this contribution we define a calculus for contracts along with a subcontract relation, and we formalize the relationship between contracts and processes (that is clients and services) exposing a given contract. Contracts are made of actions to be interpreted as either message types or communication ports. Actions may be combined by means of two choice operators: $+$ represents the *external choice*, meaning that the interacting part decides which one of alternative conversations to carry on; $\oplus$ represents the *internal choice*, meaning that the choice is not left to the interacting part. As a matter of facts, contracts are behavioral types of processes that do not manifest internal moves and the parallel structure. They are *acceptance trees* in Hennessy's terminology [11, 12].

Then we devise a *subcontract relation* $\preceq$ such that a contract σ is a subcontract of σ' if σ manifests less interacting capabilities than σ'. The subcontract relation can then be used for querying (Web services) repositories. A query for a service with contract σ may safely return services with contract σ' such that $\sigma \preceq \sigma'$. It is possible that interaction with a service that exposes a contract that is bigger than the client requires may result into unused capabilities on the server side. We argue that this is safe, because we are interested in the client's ability to complete the interaction. Such client completion property inspires a relationship between client contracts and service ones – the *contract completion* – that may be defined in terms of $\preceq$ and an appropriate complement operation over contracts.

To illustrate our contracts at work we consider a recursion-free fragment of the Calculus of Communicating Systems (CCS [13]). We define a *compliance* relation between processes such that a process – the client – interacting with another – the service – is guaranteed to *complete*. For instance the clients $(a.b \mid \overline{a}) \setminus a$ and $(a.b \mid a.c \mid \overline{a}) \setminus a$ respectively comply with the services $\overline{b}$ and $\overline{b} \mid \overline{c}$; the two clients do not comply with $\overline{c}$. We then extrapolate a contract out of a process by means of a type system defined using the expansion theorem in [15]. For instance, we are able to deduce $a.b \mid \overline{a} \vdash (a.(b.\overline{a} + \overline{a}.b) + \overline{a}.(a.b + b.a) + b) \oplus b$. Finally we prove our main result: *if the contract of a client complies with the contract of a service, then the client complies with the service.*

The expressiveness of our contract language is gauge by encoding WSDL message exchange patterns and some WSCL conversations into our contract language. Because of the $\preceq$ relation between contracts, we are able to draw some interesting considerations about similar exchange patterns, and order them according to the client's need. As we consider the recursion-free fragment of CCS, we are not able to deal with cyclic WSCL conversations, but we point out in the conclusions that their support requires well-known extensions to the contract language and to the subcontract relation.

Related work. This research was inspired by "CCS without τ" [15] and by Hennessy's model of acceptance trees [11, 12]. In facts, our contracts are an alternative representation of finite acceptance trees. While the use of formal models to describe communication protocols is not new (see for instance the exchange patterns in SSDL [19], which are based on CSP and the π-calculus), to the best of our knowledge the subcontract relation $\preceq$ is original. It is incomparable with may testing preorder and it is less discriminating than the must testing preorder [14]. The stuck free conformance relation in [10], which is inspired by the theory of refusal testing [16], is also more demanding than our subcontract relation. For instance $\mathbf{0}$ is not related with a in [10] whilst $\mathbf{0} \preceq a$.

It is worth noticing that both must testing and stuck free conformance are preserved by any CCS context without $+$ thus allowing modular refinement. This is not true for $\preceq$. For instance $a \preceq a + b$ so one might think that a service with contract a can be replaced by a service with contract $a + b$ in any context. However, the context $C = \overline{b} \mid b.\overline{a} \mid [\,]$ distinguishes the two services ($a+b$ can get stuck while a cannot). The point is that the context C, representing a client, does not comply with a, since it performs the actions b and $\overline{b}$ which are not allowed by the contract a.

Structure of the paper. In Section 2 we formally define our language for contracts along with *subcontract* and *compliance* relations. In Section 3 we relate the language with existing technologies to specify service protocols. Our notion of compliance between contracts is lifted to a notion of *compliance* between processes in Section 4. Section 5 draws our conclusion and hints to future work.

2 The contract language

The syntax of contracts uses an infinite set of *names* $\mathcal{N}$ ranged over by a, b, c, ..., and a disjoint set of *co-names* $\overline{\mathcal{N}}$ ranged over by $\overline{a}$, $\overline{b}$, $\overline{c}$, ... We let $\overline{\overline{a}} = a$. Contracts σ are defined by the following grammar:

$$
\begin{array}{lll}
\sigma \ ::= & & \textbf{contracts} \\
 & \mathbf{0} & (\textit{void}) \\
 & a.\sigma & (\textit{input prefix}) \\
 & \overline{a}.\sigma & (\textit{output prefix}) \\
 & \sigma + \sigma & (\textit{external choice}) \\
 & \sigma \oplus \sigma & (\textit{internal choice})
\end{array}
$$

Contracts are abstract definitions of conversation protocols between communicating parties. The contract $\mathbf{0}$ defines the empty conversation; the input prefix $a.\sigma$ defines a conversation protocol whose initial activity is to accept a message on the name a – representing URIs – and continuing as σ; the output prefix $\overline{a}.\sigma$ defines a conversation protocol whose initial activity is to send a message to the name a and continuing as σ. Contracts $\sigma + \sigma'$ and $\sigma \oplus \sigma'$ define conversation protocols that follow either the conversation σ or σ'; in the former ones the choice is left to the remote party, in the latter ones the choice being made locally. For example, $\texttt{Login}.(\overline{\texttt{Continue}} + \overline{\texttt{End}})$ describes the conversation protocol of a service that is ready to accept `Login`s and will `Continue` or `End` the conversation according to client's request. This contract is different from $\texttt{Login}.(\overline{\texttt{Continue}} \oplus \overline{\texttt{End}})$ where the decision whether to continue or to end is taken by the service.

In the rest of the paper, the trailing $\mathbf{0}$ is always omitted, α is used to range over names and co-names, and $\sum_{i\in 1..n}\sigma_i$ and $\bigoplus_{i\in 1..n}\sigma_i$ abbreviate $\sigma_1+\cdots+\sigma_n$ and $\sigma_1\oplus\cdots\oplus\sigma_n$, respectively. The *language* of σ, written $\mathcal{L}(\sigma)$, is the set of strings on names and co-names inductively defined as follows:

$$\begin{aligned}\mathcal{L}(\mathbf{0}) &= \{\varepsilon\}\\ \mathcal{L}(\alpha.\sigma) &= \{\alpha s \mid s\in\mathcal{L}(\sigma)\}\\ \mathcal{L}(\sigma_1+\sigma_2)=\mathcal{L}(\sigma_1\oplus\sigma_2) &= \mathcal{L}(\sigma_1)\cup\mathcal{L}(\sigma_2)\end{aligned}$$

2.1 Subcontract relation and dual contracts

Contracts retain an obvious compatibility relation that relates the conversation protocols of two communicating parties: a contract σ of a party *complies with* σ' of another party if the corresponding protocols match when they interact. Such a definition of subcontract would require the notions of communicating party, which is a process, and of contract exposed by it. We partially explore this direction in Section 4; here we give a direct definition by sticking to a structured operational semantics style. We begin by defining two notions that are preliminary to compliance: *subcontract* and *dual contract.*

Definition 1 (Transition). *Let* $\sigma \overset{\alpha}{\not\longmapsto}$ *be the least relation such that*

$$\begin{array}{rl}\mathbf{0}\overset{\alpha}{\not\longmapsto} & \\ \beta.\sigma\overset{\alpha}{\not\longmapsto} & \text{if } \alpha\neq\beta\\ \sigma\oplus\sigma'\overset{\alpha}{\not\longmapsto} & \text{if } \sigma\overset{\alpha}{\not\longmapsto} \text{ and } \sigma'\overset{\alpha}{\not\longmapsto}\\ \sigma+\sigma'\overset{\alpha}{\not\longmapsto} & \text{if } \sigma\overset{\alpha}{\not\longmapsto} \text{ and } \sigma'\overset{\alpha}{\not\longmapsto}\end{array}$$

The transition relation *of contracts, noted* $\overset{\alpha}{\longmapsto}$, *is the least relation satisfying the rules:*

$$\alpha.\sigma\overset{\alpha}{\longmapsto}\sigma$$

$$\frac{\sigma_1\overset{\alpha}{\longmapsto}\sigma_1' \quad \sigma_2\overset{\alpha}{\longmapsto}\sigma_2'}{\sigma_1+\sigma_2\overset{\alpha}{\longmapsto}\sigma_1'\oplus\sigma_2'} \qquad \frac{\sigma_1\overset{\alpha}{\longmapsto}\sigma_1' \quad \sigma_2\overset{\alpha}{\not\longmapsto}}{\sigma_1+\sigma_2\overset{\alpha}{\longmapsto}\sigma_1'}$$

$$\frac{\sigma_1\overset{\alpha}{\longmapsto}\sigma_1' \quad \sigma_2\overset{\alpha}{\longmapsto}\sigma_2'}{\sigma_1\oplus\sigma_2\overset{\alpha}{\longmapsto}\sigma_1'\oplus\sigma_2'} \qquad \frac{\sigma_1\overset{\alpha}{\longmapsto}\sigma_1' \quad \sigma_2\overset{\alpha}{\not\longmapsto}}{\sigma_1\oplus\sigma_2\overset{\alpha}{\longmapsto}\sigma_1'}$$

and closed under mirror cases for external and internal choices. We write $\sigma\overset{\alpha}{\longmapsto}$ *if there exists* σ' *such that* $\sigma\overset{\alpha}{\longmapsto}\sigma'$.

The relation $\overset{\alpha}{\longmapsto}$ is different from standard transition relations for CCS processes [13]. For example, there is always at most one contract σ' such that $\sigma\overset{\alpha}{\longmapsto}\sigma'$, while this is not the case in CCS (the process $a.b+a.c$ has two different a-successor states: b and c). This mismatch is due to the fact that contract transitions define the evolution of conversation protocols *from the perspective of the communicating parties.* Thus $a.b+a.c\overset{a}{\longmapsto}b\oplus c$ because, once the activity

a has been done, the communicating party is not aware of which conversation path has been chosen. On the contrary, CCS transitions define the evolution of processes *from the perspective of the process itself.*

We write $\sigma(\alpha)$ for the unique continuation of σ after α, that is the contract σ' such that $\sigma \overset{\alpha}{\longmapsto} \sigma'$.

Definition 2 (Ready sets and subcontracts). *Let* R *range over finite sets of names and co-names, called* ready sets*.*

$\sigma \Downarrow \mathrm{R}$ *is the least relation such that:*

$$\begin{array}{ll}
\mathbf{0} \Downarrow \emptyset & \\
\alpha.\sigma \Downarrow \{\alpha\} & \\
(\sigma + \sigma') \Downarrow \mathrm{R} \cup \mathrm{R}' & \textit{if } \sigma \Downarrow \mathrm{R} \textit{ and } \sigma' \Downarrow \mathrm{R}' \\
(\sigma \oplus \sigma') \Downarrow \mathrm{R} & \textit{if either } \sigma \Downarrow \mathrm{R} \textit{ or } \sigma' \Downarrow \mathrm{R}
\end{array}$$

The subcontract relation $\preceq$ *is the largest relation such that* $\sigma_1 \preceq \sigma_2$ *implies:*

1. *if* $\sigma_2 \Downarrow \mathrm{R}_2$ *then* $\sigma_1 \Downarrow \mathrm{R}_1$ *with* $\mathrm{R}_1 \subseteq \mathrm{R}_2$,
2. *if* $\sigma_1 \overset{\alpha}{\longmapsto} \sigma_1'$ *and* $\sigma_2 \overset{\alpha}{\longmapsto} \sigma_2'$ *then* $\sigma_1' \preceq \sigma_2'$.

Let $\sigma_1 \simeq \sigma_2$*, called* contract compatibility*, if both* $\sigma_1 \preceq \sigma_2$ *and* $\sigma_2 \preceq \sigma_1$.

The relation $\sigma \preceq \sigma'$ verifies whether the external non-determinism of σ' is greater than the external non-determinism of σ and that this holds for every α-successor of σ and σ', *provided both have such successors.* For example $a.(b \oplus c) \simeq a.b + a.c \simeq a.b \oplus a.c$ and $a.b \oplus b \preceq b$ and $b \preceq b + a.c$. It is worth to remark that $\preceq$ is not transitive: the last two relations *do not entail* $a.b \oplus b \preceq b + a.c$, which is false. This transitivity failure is not very problematic because σ and σ' are intended to play different roles in $\sigma \preceq \sigma'$, as detailed by the compliance relation. However, transitivity of $\preceq$ holds under lightweight conditions.

Proposition 1. *If* $\sigma_1 \preceq \sigma_2$ *and* $\sigma_2 \preceq \sigma_3$ *and either* $\mathcal{L}(\sigma_1) \subseteq \mathcal{L}(\sigma_2)$ *or* $\mathcal{L}(\sigma_3) \subseteq \mathcal{L}(\sigma_2)$*, then* $\sigma_1 \preceq \sigma_3$.

The relation $\preceq$ is incomparable with may testing semantics [12]: we have $a \oplus \mathbf{0} \preceq b$, while these two processes are unrelated by may testing; conversely, $a \oplus b$ and $a + b$ are may-testing equivalent, while $a + b \not\preceq a \oplus b$. The relation $\preceq$ is less discriminating than must testing semantics [12]: a and $a + b$ are unrelated in must testing while $a \preceq a + b$.

The notion of *dual contract* is used to revert the capabilities of conversation protocols. Informally, the dual contract is obtained by reverting actions with co-actions, $+$ with $\oplus$, and conversely. For example the dual contract of $a \oplus \overline{b}$ is $\overline{a} + b$. However, this naïve transformation is fallible because in the contract language some external choices are actually internal choices in disguise. For example, $a.b + a.c \simeq a.(b \oplus c)$ but their dual contracts are respectively $\overline{a}.\overline{b} \oplus \overline{a}.\overline{c}$ and $\overline{a}.(\overline{b} + \overline{c})$, and they tell very different things. In the first one, the communicating party cannot decide which action to perform after $\overline{a}$, whereas this possibility is granted in the second one. To avoid such misbehavior, we define dual contracts on contracts in normal form. We use the same forms introduced in [12]. Let the *normed contract* of σ, noted $\mathtt{nc}(\sigma)$, be

$$\mathtt{nc}(\sigma) \stackrel{\text{def}}{=} \bigoplus_{\sigma \Downarrow \mathrm{R}} \sum_{\alpha \in \mathrm{R}} \alpha.\mathtt{nc}(\sigma(\alpha)) \ .$$

For example

$$\begin{aligned}\mathtt{nc}((a.b \oplus b.c) + (a.b.d \oplus c.b)) = \quad & a.b.(\mathbf{0} \oplus d) \\ & \oplus\ (a.b.(\mathbf{0} \oplus d) + c.b) \\ & \oplus\ (a.b.(\mathbf{0} \oplus d) + b.c) \\ & \oplus\ (b.c + c.b)\end{aligned}$$

Lemma 1. *$\sigma \simeq \mathtt{nc}(\sigma)$ and $\mathcal{L}(\sigma) = \mathcal{L}(\mathtt{nc}(\sigma))$.*

Definition 3 (Dual contracts). *The* dual contract *of σ, noted $\overline{\sigma}$, is defined as*

$$\overline{\sigma} \stackrel{\text{def}}{=} \sum_{\sigma \Downarrow \mathrm{R}} \bigoplus_{\alpha \in \mathrm{R}} \overline{\alpha}.\overline{\sigma(\alpha)}$$

where, by convention, we have $\bigoplus_{\sigma \in \emptyset} \sigma = \mathbf{0}$.

The dual operator is not contravariant with respect to $\preceq$. For example, $a \preceq a.b$, but $\overline{a.b} = \overline{a}.\overline{b} \not\preceq \overline{a}$. For similar reasons, contract compatibility is not preserved. For example, $\mathbf{0} \simeq \mathbf{0} \oplus a$ but $\overline{\mathbf{0}} = \mathbf{0} \not\simeq \mathbf{0} + \overline{a} = \overline{\mathbf{0} \oplus a}$. However a limited form of contravariance, which will result fundamental in the following, is satisfied by the dual operator.

Lemma 2. *$\overline{\sigma} \preceq \overline{\sigma \oplus \sigma'}$.*

2.2 Contract compliance

Every preliminary notion has been set for the definition of contract compliance.

Definition 4 (Contract compliance). *A contract σ* complies with *σ', noted $\sigma \ll \sigma'$, if and only if $\overline{\sigma} \preceq \sigma'$.*

The notion of contract compliance is meant to be used for querying a Web service repository. A client with contract σ will interact successfully with every service with contract σ' provided $\sigma \ll \sigma'$. For example, consider a client whose conversation protocol states that it intends to choose whether to be notified either on a name a or on a name b. Its contract might be $a \oplus b$. Querying a repository for compliant services means returning every service whose conversation protocol is $\overline{a} + \overline{b}$, or $\overline{a} + \overline{b} + a$, or $\overline{a}.c + \overline{b}$, etc. The guarantee that we provide (see Section 4) is that, whatever service returned by the repository is chosen, the client will conclude his conversation. This asymmetry between the left hand side of $\preceq$ (and of $\ll$) and the right hand side is the reason of the failure of transitivity. More precisely, in $a.b \oplus b \preceq b$ and in $b \preceq a.c + b$, we are guaranteeing the termination of clients manifesting the two left hand sides contracts with respect to services manifesting the two right hand side contracts. This property is not transitive.

3 On the expressive power of the contract language

In this section we relate our contract language to existing technologies for specifying service protocols.

3.1 Message exchange patterns in WSDL

The Web Service Description Language (WSDL) Version 1.1 [6] permits to describe and publish abstract and concrete descriptions of Web services. Such descriptions include the schema [9] of messages exchanged between client and server, the name and type of *operations* that the service exposes, as well as the locations (URLs) where the service can be contacted. In addition, it defines four interaction patterns determining the order and direction of exchanged messages. For instance, the *request-response* pattern is used to describe a synchronous operation where the client issues a request and subsequently receives a response from the service.

The second version of WSDL [3–5] allows users to agree on message exchange patterns (MEP) by specifying in the required `pattern` attribute of operation elements an absolute URI that identifies the MEP. It is important to notice that these URIs act as global identifiers (their content is not important) for MEPs, whose semantics is usually given in plain English. In particular, WSDL 2.0 [4] predefines four message exchange patterns (each pattern being uniquely identified by a different URI) for describing services where the interaction is initiated by clients. Let us shortly discuss how the informal plain English semantics of these patterns can be formally defined in our contract language. Consider the WSDL 2.0 fragment

```
<operation name="A" pattern="http://www.w3.org/2006/01/wsdl/in-only">
  <input messageLabel="In"/>
</operation>
<operation name="B"
           pattern="http://www.w3.org/2006/01/wsdl/robust-in-only">
  <input messageLabel="In"/>
  <outfault messageLabel="Fault"/>
</operation>
<operation name="C" pattern="http://www.w3.org/2006/01/wsdl/in-out">
  <input messageLabel="In"/>
  <output messageLabel="Out"/>
  <outfault messageLabel="Fault"/>
</operation>
<operation name="D" pattern="http://www.w3.org/2006/01/wsdl/in-opt-out">
  <input messageLabel="In"/>
  <output messageLabel="Out"/>
  <outfault messageLabel="Fault"/>
</operation>
```

which defines four operations named `A`, `B`, `C`, and `D`. The first two operations are *asynchronous* by accepting only an incoming message labeled `In`. The last two operations are *synchronous* by accepting an incoming message labeled `In` and replying with a message labeled `Out`. In the `B` operation a fault message can occur after the input. The `C` operation always produces an output message (see `in-out` in its `pattern` attribute), unless a fault occurs. In the `D` operation the reply is optional, as stated by the `in-opt-out` exchange pattern attribute, and again it may fail with `Fault`.

We can encode the contract of the pattern of the `A` operation in our contract language as $\mathtt{inOnly} = \mathtt{In}.\overline{\mathtt{End}}$, that is an input action representing the client's request followed by a message $\overline{\mathtt{End}}$ that is sent from the service to notify the client that the interaction has completed.

The `B` operation can be encoded as

$$\mathtt{robustInOnly} = \mathtt{In}.(\overline{\mathtt{End}} \oplus \overline{\mathtt{Fault}}.\overline{\mathtt{End}})$$

where after the client's request, the interaction may follow two paths, representing successful and faulty computations respectively. In the former case the end of the interaction is immediately signaled to the client. In the latter case a message `Fault` is sent to the client, followed by `End`. The use of the internal choice for combining the two paths states that it is the service that decides whether the interaction is successful or not. This means that a client compliant with this service can either stop after the request or it must be able to handle both the `End` and `Fault` messages: the omission of handling, say, `Fault` would result into an uncaught exception.

The need for an explicit `End` message to signal a terminated interaction is not immediately evident. In principle, the optional fault message could have been encoded as $\mathtt{In}.(\mathbf{0} \oplus \overline{\mathtt{Fault}})$. A client compliant with this service must be able to receive and handle the `Fault` message, but it must also be able to complete the interaction without further communication from the service. The point is that the client cannot distinguish a completed interaction where the service has internally decided to behave like $\mathbf{0}$ from an interaction where the service has internally decided to behave like `Fault`, but it is taking a long time to respond. By providing an explicit `End` message signaling a completed interaction, the service tells the client not to wait for further messages. By this reasoning, the `End` message after `Fault` is not strictly necessary, but we write it for uniformity.

By similar arguments the contract of the `C` operation can be encoded as

$$\mathtt{inOut} = \mathtt{In}.(\overline{\mathtt{Out}}.\overline{\mathtt{End}} \oplus \overline{\mathtt{Fault}}.\overline{\mathtt{End}})$$

and the contract of the `D` operation as

$$\mathtt{inOptOut} = \mathtt{In}.(\overline{\mathtt{End}} \oplus \overline{\mathtt{Out}}.\overline{\mathtt{End}} \oplus \overline{\mathtt{Fault}}.\overline{\mathtt{End}})$$

It is worth noticing how these contracts are ordered according to our definition of $\preceq$. We have $\mathtt{inOptOut} \preceq \mathtt{robustInOnly}$ and $\mathtt{robustInOnly} \preceq \mathtt{inOnly}$. Indeed, a client compliant with `inOptOut` must be able to complete immediately after the request, but it is also able to handle a `Out` message and a `Fault` message. The `robustInOnly` can only produce an `End` message or a `Fault` message, hence it is "more deterministic" than `inOptOut`. Similarly, `inOnly` is more deterministic than `robustInOnly` since it can only send an `End` message after the client's request. Finally, note that `inOptOut` $\preceq$ `inOut` also holds.

3.2 Conversations in WSCL

The WSDL message exchange patterns cover only the simplest forms of interaction between a client and a service. More involved forms of interactions, in particular stateful interactions, cannot be captured if not as informal annotation within the WSDL interface. The Web service conversation language WSCL [1] provides a more general specification language for describing complex *conversations* between two communicating parties, by means of an activity diagram. The diagram is basically made of *interactions* which are connected with each other by means of *transitions*. An interaction is a basic one-way or two-way communication between the client and the server. Two-way communications are just a shorthand for two sequential one-way interactions. Each interaction has a *name* and a list of *document types* that can be exchanged during its execution. A transition connects a *source* interaction with a *destination* interaction. A transition may be *labeled* by a document type if it is active only when a message of that specific document type was exchanged during the previous interaction.

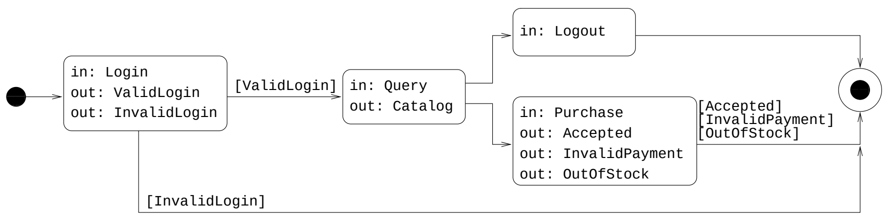

Fig. 1. Contract of a simple e-commerce service as a WSCL diagram.

Below we encode the contract σ of a simplified e-commerce service (Figure 1) where the client is required to login before it can issue a query and thus receive a catalog. From this point on, the client can decide whether to purchase an item from the catalog or to logout and leave. In case of purchase, the service may either report that the purchase was successful, or that the item is out-of-stock, or that the client's payment was refused:

$$\begin{aligned}\sigma \stackrel{\text{def}}{=} \ & \mathtt{Login}.(\overline{\mathtt{InvalidLogin}}.\overline{\mathtt{End}} \oplus \overline{\mathtt{ValidLogin}}.\mathtt{Query}.\overline{\mathtt{Catalog}}.(\\ & \quad \mathtt{Logout}.\overline{\mathtt{End}} + \mathtt{Purchase}.(\\ & \qquad \overline{\mathtt{Accepted}}.\overline{\mathtt{End}} \oplus \overline{\mathtt{InvalidPayment}}.\overline{\mathtt{End}} \oplus \overline{\mathtt{OutOfStock}}.\overline{\mathtt{End}})))\end{aligned}$$

Notice that unlabeled transitions in Figure 1 correspond to external choices in σ, whereas labeled transitions correspond to internal choices. It is also interesting to notice that WSCL explicitly accounts for a termination message (called "empty" in the WSCL specification, the final interaction on the right end in Figure 1) that is used for modeling the end of a conversation. The presence of this termination message finds a natural justification in our formal contract language, as explained above.

Now assume that the service is extended with a booking capability, so that after looking at the catalog the client may book an item to be bought at some later time. The contract of the service would change to σ' as follows:

$$\sigma' \stackrel{\text{def}}{=} \dots \texttt{Logout}.\overline{\texttt{End}} + \texttt{Book}.\overline{\texttt{End}} + \texttt{Purchase}.(\dots)$$

We notice that $\sigma \preceq \sigma'$ and $\mathcal{L}(\sigma) \subseteq \mathcal{L}(\sigma')$, that is σ' offers more capabilities than σ.

4 Compliance

Compliance relates a client process with a service process. A client is compliant with a service if the client terminates (i.e. it has no more interactions to perform) for every possible interaction with the service. That is, compliance induces a *completion property* for the client but not for the service. In order to formalize compliance we define processes and their dynamics. Then we demonstrate that it is possible to associate a contract to a process such that (process) compliance follows by the compliance of the corresponding contracts.

In this contribution, processes are finite CCS terms. The extension to CCS terms is not trivial and left for future work. For the sake of simplicity we do not include choice and relabeling operators. The transition relation is standard; therefore we omit comments.

Definition 5. Processes P *are defined by the following grammar:*

$$P \ ::= \ \mathbf{0} \ \mid \ a.P \ \mid \ \overline{a}.P \ \mid \ P \setminus a \ \mid \ P \,|\, P$$

Let μ range over $\mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$. The transition relation *of processes, noted $\xrightarrow{\mu}$, is the least relation satisfying the rules:*

$$\text{(IN)}\ \ a.P \xrightarrow{a} P \qquad \text{(OUT)}\ \ \overline{a}.P \xrightarrow{\overline{a}} P \qquad \text{(RES)}\ \ \frac{P \xrightarrow{\mu} Q \quad \mu \notin \{a, \overline{a}\}}{P \setminus a \xrightarrow{\mu} Q \setminus a}$$

$$\text{(PAR)}\ \ \frac{P \xrightarrow{\mu} Q}{P \,|\, R \xrightarrow{\mu} Q \,|\, R} \qquad \text{(COM)}\ \ \frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\overline{\alpha}} Q'}{P \,|\, Q \xrightarrow{\tau} P' \,|\, Q'}$$

The transitions of $P \,|\, Q$ have mirror cases that have been omitted.

We write $\stackrel{\tau}{\Longrightarrow}$ for $\xrightarrow{\tau}^{*}$ and $\stackrel{\alpha}{\Longrightarrow}$ for $\xrightarrow{\tau}^{*}\xrightarrow{\alpha}\xrightarrow{\tau}^{*}$.

The compliance of a client process with a service is defined as follows.

Definition 6 (Compliance). *Let $P \parallel Q \longrightarrow P' \parallel Q'$ be the least relation such that:*

- *if $P \xrightarrow{\tau} P'$ then $P \parallel Q \longrightarrow P' \parallel Q$;*
- *if $Q \xrightarrow{\tau} Q'$ then $P \parallel Q \longrightarrow P \parallel Q'$;*
- *if $P \xrightarrow{\alpha} P'$ and $Q \xrightarrow{\overline{\alpha}} Q'$ then $P \parallel Q \longrightarrow P' \parallel Q'$.*

Let $P \ll Q$, read P complies with *Q, if one of the following holds:*

1. *$P \overset{\alpha}{\not\longrightarrow}$, or*
2. *$P \parallel Q \longrightarrow P' \parallel Q'$ and $P' \ll Q'$.*

Process compliance has been noted in the same way as contract compliance in Section 2. This abuse is justified because the two notions are strongly related, as we will prove shortly.

Processes expose (principal) contracts. This is defined by an inference system that uses two auxiliary operators over contracts:

1. $\sigma \setminus a$ is defined by induction on the structure of σ:

$$\begin{array}{rcl} \mathbf{0} \setminus a & = & \mathbf{0} \\ (\alpha.\sigma) \setminus a & = & \begin{cases} \mathbf{0} & \text{if } \alpha \in \{a, \overline{a}\} \\ \alpha.(\sigma \setminus a) & \text{otherwise} \end{cases} \\ (\sigma + \sigma') \setminus a & = & \sigma \setminus a + \sigma' \setminus a \\ (\sigma \oplus \sigma') \setminus a & = & \sigma \setminus a \oplus \sigma' \setminus a \end{array}$$

2. The operator "$\,|\,$" is commutative with $\mathbf{0}$ as identity, such that $\sigma \,|\, (\sigma' \oplus \sigma'') = (\sigma \,|\, \sigma') \oplus (\sigma \,|\, \sigma'')$, and $\sigma \,|\, (\sigma' + (\sigma'' \oplus \sigma''')) = \sigma \,|\, ((\sigma' + \sigma'') \oplus (\sigma' + \sigma'''))$. This allows us to define $\sigma \,|\, \sigma'$ when σ and σ' are external choices of prefixes. Our definition corresponds to the *expansion law* in [15]. Let $\sigma = \sum_{i \in I} \alpha_i.\sigma_i$ and $\sigma' = \sum_{j \in J} \alpha'_j.\sigma'_j$, then

$$\sigma \,|\, \sigma' \overset{\text{def}}{=} \begin{cases} \sum_{i\in I} \alpha_i.(\sigma_i \,|\, \sigma') \;+\; \sum_{j \in J} \alpha'_j.(\sigma \,|\, \sigma'_j) \\ \hfill \text{if } \alpha_i \neq \overline{\alpha'_j} \text{ for every } i \in I,\ j \in J \\ \\ \Big(\sum_{i\in I} \alpha_i.(\sigma_i \,|\, \sigma') \;+\; \sum_{j \in J} \alpha'_j.(\sigma \,|\, \sigma'_j) \;+\; \bigoplus_{\alpha_i = \overline{\alpha'_j}} (\sigma_i \,|\, \sigma'_j)\Big) \\ \quad \oplus \bigoplus_{\alpha_i = \overline{\alpha'_j}} (\sigma_i \,|\, \sigma'_j) \hfill \text{otherwise} \end{cases}$$

Definition 7. *Let $P \vdash \sigma$ be the least relation such that*

$$\mathbf{0} \vdash \mathbf{0} \qquad \frac{P \vdash \sigma}{a.P \vdash a.\sigma} \qquad \frac{P \vdash \sigma}{\overline{a}.P \vdash \overline{a}.\sigma} \qquad \frac{P \vdash \sigma}{P \setminus a \vdash \sigma \setminus a} \qquad \frac{P \vdash \sigma \quad Q \vdash \sigma'}{P \,|\, Q \vdash \sigma \,|\, \sigma'}$$

As anticipated, compliance of processes may be inferred from compliance of the corresponding contracts. This property, formalized in Theorem 1, requires few preliminary statements.

Lemma 3. *Let $P \vdash \sigma$, $P \xrightarrow{\mu} P'$, and $P' \vdash \sigma'$*

(a) *if $\mu = \tau$ then $\sigma \preceq \sigma'$, $\overline{\sigma'} \preceq \overline{\sigma}$, and $\mathcal{L}(\sigma') \subseteq \mathcal{L}(\sigma)$;*
(b) *if $\mu = \alpha$ then $\sigma(\alpha) \preceq \sigma'$, $\overline{\sigma'} \preceq \overline{\sigma(\alpha)}$, and $\mathcal{L}(\sigma') \subseteq \mathcal{L}(\sigma(\alpha))$.*

Proof. (Sketch) We proceed by induction on the derivation of $P \xrightarrow{\mu} P'$.

The base case corresponds to the application of either (IN) or (OUT). Since P has the form $\alpha.P'$ we have $\sigma(\alpha) = \sigma'$. Therefore we conclude $\sigma(\alpha) \preceq \sigma'$, $\overline{\sigma'} \preceq \overline{\sigma(\alpha)}$, and $\mathcal{L}(\sigma') = \mathcal{L}(\sigma(\alpha))$.

In the inductive case there are several sub-cases corresponding to the last rule that has been applied. We discuss (COM) and (PAR).

- (COM) implies $P = Q \mid R$ with $Q \xrightarrow{\alpha} Q'$ and $R \xrightarrow{\overline{\alpha}} R'$. Let $Q \vdash \sigma_1$, $Q' \vdash \sigma_1'$, $R \vdash \sigma_2$, and $R' \vdash \sigma_2'$. By definition of "$\mid$", we have $\sigma_1 \mid \sigma_2 = \bigoplus_{i\in I} \sigma_i''$ with $\sigma_j'' = \sigma_1' \mid \sigma_2'$ for some $j \in I$. Hence $\sigma_1 \mid \sigma_2 \preceq \sigma_1' \mid \sigma_2'$ follows by definition of $\preceq$ and $\overline{\sigma_1' \mid \sigma_2'} \preceq \overline{\sigma_1 \mid \sigma_2}$ follows by Lemma 2. It remains to show $\mathcal{L}(\sigma_1' \mid \sigma_2') \subseteq \mathcal{L}(\sigma_1 \mid \sigma_2)$. This is a straightforward consequence of the definition of "$\mid$" and $\mathcal{L}(\cdot)$.
- (PAR) implies $P = Q \mid R$ with $Q \xrightarrow{\mu} Q'$ and $Q \vdash \sigma_1$, $R \vdash \sigma_2$, and $Q' \vdash \sigma_1'$.
  - If $\mu = \tau$, by definition of "$\mid$", we have $\sigma_1 = \bigoplus_{i\in I} \sigma_i''$ with $\sigma_j'' = \sigma_1'$ for some $j \in I$. Then $\sigma_1 \mid \sigma_2 = (\bigoplus_{i\in I} \sigma_i'') \mid \sigma_2 = \bigoplus_{i\in I} (\sigma_i'' \mid \sigma_2)$ and $\sigma_1 \mid \sigma_2 \preceq \sigma_1' \mid \sigma_2$ follows by definition of $\preceq$ while $\overline{\sigma_1' \mid \sigma_2} \preceq \overline{\sigma_1 \mid \sigma_2}$ follows by Lemma 2. By definition of $\mathcal{L}(\cdot)$ we also conclude that $\mathcal{L}(\sigma_1' \mid \sigma_2) \subseteq \mathcal{L}(\sigma_1 \mid \sigma_2)$.
  - If $\mu = \alpha$, by the inductive hypothesis we have $\sigma_1(\alpha) \preceq \sigma_1'$ and $\overline{\sigma_1'} \preceq \overline{\sigma(\alpha)}$. Since $Q \xrightarrow{\alpha} Q'$, by definition of "$\mid$" we have that $\sigma_1 \mid \sigma_2$ has the shape $\rho_1 \oplus (\rho_2 + \alpha.(\sigma_1' \mid \sigma_2) + \rho_3) \oplus \rho_4$ where an arbitrary number of the ρ_i's may be missing. Hence $(\sigma_1 \mid \sigma_2)(\alpha) = \cdots \oplus (\sigma_1' \mid \sigma_2) \oplus \cdots$. Then $(\sigma_1 \mid \sigma_2)(\alpha) \preceq \sigma_1' \mid \sigma_2$ follows by definition of $\preceq$ and $\overline{\sigma_1' \mid \sigma_2} \preceq \overline{(\sigma_1 \mid \sigma_2)(\alpha)}$ by Lemma 2. By definition of $\mathcal{L}(\cdot)$ we also conclude that $\mathcal{L}(\sigma_1' \mid \sigma_2) \subseteq \mathcal{L}((\sigma_1 \mid \sigma_2)(\alpha))$.

$\square$

Theorem 1. *If $P \vdash \sigma$, $Q \vdash \sigma'$, and $\sigma \ll \sigma'$ then $P \ll Q$.*

Proof. A maximal computation of the system $P \parallel Q$ is a sequence of systems $P_1 \parallel Q_1, \dots, P_n \parallel Q_n$ such that $P_1 = P$, $Q_1 = Q$, for every $i = \{1, \dots, n-1\}$ we have $P_i \parallel Q_i \longrightarrow P_{i+1} \parallel Q_{i+1}$, and $P_n \parallel Q_n \not\longrightarrow$. The proof is by induction on n.

If $n = 0$, then $P \parallel Q \not\longrightarrow$. We have two possibilities: if $P \overset{\alpha}{\not\longrightarrow}$ then by definition $P \ll Q$. So let us suppose, by contradiction, that whenever $P \xrightarrow{\alpha}$ we have $Q \overset{\overline{\alpha}}{\not\longrightarrow}$. Since $P \vdash \sigma$ and $Q \vdash \sigma'$ this means that for any ready set R of σ there is no ready set S of σ' such that $\overline{\mathrm{R}} \cap \mathrm{S} \neq \emptyset$. From $P \xrightarrow{\alpha}$ and $P \vdash \sigma$ we know that $\sigma \Downarrow \mathrm{R}$ and $\alpha \in \mathrm{R}$ for some ready set R. That is, σ has at least one nonempty ready set. Thus, from the definition of $\overline{\sigma}$, we know that *every* ready set of $\overline{\sigma}$ is not empty. By definition of contract compliance we know that $\overline{\sigma} \preceq \sigma'$ and from the definition of $\preceq$ we have that any ready set S of σ' shares at least an action with $\overline{\mathrm{R}}$ for some ready set R of σ, which is absurd.

If $n > 0$, assume that the theorem is true for any computation of length $n-1$. We have three cases:

($P \longrightarrow P'$) Assume $P' \vdash \sigma''$, then from Lemma 3(a) we know that $\overline{\sigma''} \preceq \overline{\sigma}$ and $\mathcal{L}(\overline{\sigma''}) \subseteq \mathcal{L}(\overline{\sigma})$, hence by Proposition 1 we have $\overline{\sigma''} \preceq \sigma'$ that is $\sigma'' \ll \sigma'$. By the induction hypothesis we conclude that $P' \ll Q$ hence $P \ll Q$.

($Q \longrightarrow Q'$) Assume $Q' \vdash \sigma''$, then from Lemma 3(a) we know that $\sigma' \preceq \sigma''$ and $\mathcal{L}(\sigma'') \subseteq \mathcal{L}(\sigma')$, hence by Proposition 1 we have $\overline{\sigma} \preceq \sigma''$ that is $\sigma \ll \sigma''$. By the induction hypothesis we conclude that $P \ll Q'$ hence $P \ll Q$.

($P \xrightarrow{\alpha} P'$ **and** $Q \xrightarrow{\overline{\alpha}} Q'$) Assume that $P' \vdash \sigma''$ and $Q' \vdash \sigma'''$. From Lemma 3(b) know that $\overline{\sigma''} \preceq \overline{\sigma(\alpha)}$ and $\mathcal{L}(\overline{\sigma''}) \subseteq \mathcal{L}(\overline{\sigma(\alpha)})$, and by definition of dual contract we have $\overline{\sigma(\alpha)} = \overline{\sigma}(\overline{\alpha})$. Again from Lemma 3(b) we know that $\sigma'(\alpha) \preceq \sigma'''$ and $\mathcal{L}(\sigma''') \subseteq \mathcal{L}(\sigma'(\alpha))$. By Proposition 1 we have $\overline{\sigma''} \preceq \sigma'''$ that is $\sigma'' \ll \sigma'''$. The computation starting from $P' \parallel Q'$ has length $n-1$, by the induction hypothesis we have $P' \ll Q'$ so we conclude $P \ll Q$. □

5 Conclusion and future work

In this paper we have started an investigation aimed at the definition of a formal contract language suitable for describing interactions of clients with Web services. We have defined a precise notion of compatibility between services, called subcontract relation, so that equivalent services can be safely replaced with each other. This notion of compatibility is immediately applicable in any query-based system for service discovery, as well as for verifying that a service implementation respects its interface. To the best of our knowledge, this relation is original and it does not coincide with either must, or may, or testing preorders. Based on the subcontract relation, we have provided a formal notion of compliance, such that clients that are verified to be compliant with a contract are guaranteed to successfully complete the interaction with any service that exports that contract.

We have based our investigation on a very simple model of concurrency, the Calculus of Communicating Systems [13] without recursion, since this is but the first step of our investigation. Starting from this basis, we plan to pursue several lines of research. First and foremost we want to explore whether it is possible to modify our subcontract relation so that it is transitive, while preserving its main properties. The lack of transitivity has a non negligible impact on the use our relation. For instance, while it is possible to replace a given service with a new service whose subcontract is greater than the original service's contract, it is not possible to renew this operation without taking into account the original contract. After that we plan to study the addition of some form of recursion in order to model protocols whose length is not statically bound, as well as a better support of optional contracts. While these last points should not pose any particular problem, the passage from a CCS-like formalism to a π-calculus one will be much a more challenging task. Nevertheless this passage to a higher order formalism looks crucial for more than one reason. First it will allow us to take into account and generalize the forthcoming versions of WSDL. Also, it will more faithfully mimic WSCL protocols which discriminate on the content of messages. Besides, the type of these parameters could also be used to define contract isomorphisms to improve service discovery. In particular we will study *provable* isomorphisms, that is, isomorphisms for which it is possible to exhibit a process that "converts" the two contracts: for instance, imagine that we search for a service that implements the contract `In(Int).In(Int)`, that is, a service that sequentially waits twice for an integer on the port `In`; the query may return a reference to a service with a contract isomorphic to it, say, `In(Int`$\times$`Int)` together with a process that "proves" that these two contracts are isomorphic, that is, in the specific case, a process that buffers the two inputs and sends the pair of them on `In`: by composing this process with the original client (written for the first contract) one obtains a client complying with the discovered service.

On the linguistic side we would like to explore new process constructions that could take into account information available with contracts. For instance imagine a client that wants to use a service exporting the contract $(a + b) \oplus a$; in the simple language of Section 2 the client cannot specify that it wants to connect with b if available, and on a otherwise. We want also to devise query languages for service discovery, in particular we aim to devise a simple set-theoretic interpretation of contracts as sets of processes, use it to add union, intersection, and negation operators for contracts, and subsequently use these as query primitives.

A final issue brought by higher-order and whose exploration looks promising is that higher-order channels will allow us to use a continuation passing style (CPS) of programming. It is well-known that CPS can be used for stateless implementation of interactive web-sessions [17], thus we plan to transpose such a technique to contracts and resort to CPS to describe stateful interactions of services.

Acknowledgments. This work was partially funded by the ACI project "Transformation Languages for XML: Logics and Applications" (TraLaLA).

References

1. A. Banerji, C. Bartolini, D. Beringer, V. Chopella, et al. *Web Services Conversation Language (*WSCL*) 1.0*, Mar. 2002. `http://www.w3.org/TR/2002/NOTE-wscl10-20020314`.
2. D. Beringer, H. Kuno, and M. Lemon. *Using* WSCL *in a* UDDI *Registry 1.0*, 2001. UDDI Working Draft Best Practices Document, `http://xml.coverpages.org/HP-UDDI-wscl-5-16-01.pdf`.
3. D. Booth and C. K. Liu. *Web Services Description Language (*WSDL*) Version 2.0 Part 0: Primer*, Mar. 2006. `http://www.w3.org/TR/2006/CR-wsdl20-primer-20060327`.
4. R. Chinnici, H. Haas, A. A. Lewis, J.-J. Moreau, et al. *Web Services Description Language (*WSDL*) Version 2.0 Part 2: Adjuncts*, Mar. 2006. `http://www.w3.org/TR/2006/CR-wsdl20-adjuncts-20060327`.
5. R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. *Web Services Description Language (*WSDL*) Version 2.0 Part 1: Core Language*, Mar. 2006. `http://www.w3.org/TR/2006/CR-wsdl20-20060327`.

6. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (*WSDL*) 1.1*, 2001. `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`.
7. J. Colgrave and K. Januszewski. Using WSDL in a UDDI registry, version 2.0.2. Technical note, OASIS, 2004. `http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm`.
8. R. D. Cosmo. *Isomorphisms of Types: from Lambda Calculus to Information Retrieval and Language Desig.* Birkhauser, 1995. ISBN-0-8176-3763-X.
9. D. C. Fallside and P. Walmsley. *XML Schema Part 0: Primer Second Edition*, Oct. 2004. `http://www.w3.org/TR/xmlschema-0/`.
10. C. Fournet, C. A. R. Hoare, S. K. Rajamani, and J. Rehof. Stuck-free conformance. Technical Report MSR-TR-2004-69, Microsoft Research, July 2004.
11. M. Hennessy. Acceptance trees. *JACM: Journal of the ACM*, 32(4):896–928, 1985.
12. M. C. B. Hennessy. *Algebraic Theory of Processes.* Foundation of Computing. MIT Press, 1988.
13. R. Milner. *A Calculus of Communicating Systems.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.
14. R. D. Nicola and M. Hennessy. Testing equivalences for processes. *Theor. Comput. Sci*, 34:83–133, 1984.
15. R. D. Nicola and M. Hennessy. CCS without tau's. In *TAPSOFT '87/CAAP '87: Proceedings of the International Joint Conference on Theory and Practice of Software Development, Volume 1: Advanced Seminar on Foundations of Innovative Software Development I and Colloquium on Trees in Algebra and Programming*, pages 138–152, London, UK, 1987. Springer-Verlag.
16. I. Phillips. Refusal testing. *Theor. Comput. Sci.*, 50(3):241–284, 1987.
17. C. Queinnec. Inverting back the inversion of control or, continuations versus page-centric programming. *SIGPLAN Not.*, 38(2):57–64, 2003.
18. M. Rittri. Retrieving library functions by unifying types modulo linear isomorphism. *RAIRO Theoretical Informatics and Applications*, 27(6):523–540, 1993.
19. Savas Parastatidis and Jim Webber. *MEP SSDL Protocol Framework*, Apr. 2005. `http://ssdl.org`.