Augmenting Semantic Web Service Description With Compositional Specification

Monika Solanki
Software Technology
Research Laboratory
The Gateway
Leicester LE1 9BH, UK
monika@dmu.ac.uk

Antonio Cau
Software Technology
Research Laboratory
The Gateway
Leicester LE1 9BH, UK
acau@dmu.ac.uk

Hussein Zedan
Software Technology
Research Laboratory
The Gateway
Leicester LE1 9BH, UK
zedan@dmu.ac.uk

ABSTRACT

Current ontological specifications for semantically describing properties of Web services are limited to their static interface description. Normally for proving properties of service compositions, mapping input/output parameters and specifying the pre/post conditions are found to be sufficient. However, these properties are assertions only on the initial and final states of the service respectively. They do not help in specifying/verifying ongoing behaviour of an individual service or a composed system. We propose a framework for enriching semantic service descriptions with two compositional assertions: *assumption* and *commitment* that facilitate reasoning about service composition and verification of their integration. The technique is based on Interval Temporal Logic(ITL): a sound formalism for specifying and proving temporal properties of systems. Our approach utilizes the recently proposed Semantic Web Rule Language.

Categories and Subject Descriptors

D.3.2 [**Language Classifications**]: Constraint and logic languages; F.4.1 [**Mathematical Logic**]: Temporal Logic; D.3.3 [**Programming Languages**]: General—*standards*

General Terms

Languages, Theory, Verification

Keywords

Semantic Web services, Web services, Interval Temporal Logic, OWL, OWL-S, SWRL, Assumption - Commitment

1. INTRODUCTION AND MOTIVATION

Markup languages for specifications of Web services are set to play an important role in enabling dynamic service discovery and composition by human users and software agents. There is a plethora of languages proposed by academic and industrial research groups for service description, discovery and composition with XML as their backbone. The semantic web has contributed substantially via providing ontology description languages like OWL [12]. Prominent efforts can be seen in the form of OWL-S [23]: an ontology for describing profile, process and grounding models for a service. Related work is also being done in the form of IRS (Internet Reasoning Services) [19].

WWW2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-844-X/04/0005.

The Current Web service description languages however suffer from the lack of their ability to provide constructs and concepts that enable reasoning about runtime service behaviour. Languages like WSDL [21] and BPEL4WS [24] do not have the provision for specifying the conditions a service provider would want to impose on its environment to guarantee a valid service execution. Similarly there are no ways for the service provider to describe what assertions would be true once the service has been executed. OWL-S does provide concepts like preconditions and conditional effects. The WSMF [5] defines pre- and post-conditions for services. These contribute to some extent towards their behavioural description. However pre/post-conditions and effects are limited to static descriptions. They are required to hold only at the initial and final states of the service execution.

In a scenario where Web services are black boxes and the client or the service requester (human/agent) has no control over the various stages of service execution, it becomes necessary to enrich the service description with certain properties which would enable reasoning about service behaviour while it is in execution. The need for such rich specifications becomes evident while reasoning about the composition of services and verification of the composition. Current composition planners/engines [25, 22] use input/output mapping, type characteristics of these parameters and initial, final state predicates to generate compositions/plans. However, to ensure a sound composition, services need to be composed using a specification technique that characterizes ongoing behaviour of the service.

Further, since Web services abstract implementation details, composing agents (human/software) have no means to validate an ongoing composition and take appropriate measures in case the specifications cease to be satisfied. Behavioural specification of services should include assertions that not only validate their initial and final states, but also their intermediate states. These states could be critical decision making stages during computation or the states at which messages are exchanged between the services. The rationale for these additional observables become more apparent while composing services that execute concurrently as the possibility of synchronization and communication between the services make the intermediate states as important as the initial and final one. Moreover with such properties in hand, verification of the composition at runtime is reduced to checking the assertions required to hold by every service in the composition using an engine designed to handle temporal properties.

We propose a methodology to augment the semantic description of a service with temporal properties formally called *assumption*

and *commitment*[1]. The properties are specified in Interval Temporal Logic(ITL) [17, 18, 16, 2], our underlying formalism for reasoning about service behaviour over periods of time. These assertions are specified using predicates in first order logic with temporal operators only over their observable behaviour and do not depend on any additional knowledge about the underlying execution mechanism of the services. Further, we show that such assertional specifications are compositional and this strategy can be effectively applied for the verification of a composed service on the basis of the specification of the individual services. The formalism thus provides a powerful technique for reasoning about service composition, execution and runtime verification of service behaviour.

We choose OWL-S: an OWL ontology as our starting point for specifying these properties. OWL-S scores highly in expressiveness over other languages. OWL has well-defined semantics in description logic. OWL-S has operational semantics defined in Petri Nets [20] and subtype polymorphism [4]. Although other languages for Web service descriptions claim to have semantics defined in Pi-calculus/Petri nets, there are no documents supporting the same. Recently a preliminary proposal for a rule language has been put forward to express rules and constraints [9] in the framework of the OWL language. This allows us to express property predicates as an OWL ontology. In this paper we show how the compositional properties of A - C can be expressed as a SWRL ontology. This representation can be made a part of the process model in OWL-S. Our approach differs from conventional approaches as we consider validation and verification to be an integral part of service composition. This also makes it readily applicable to the execution monitoring model proposed in OWL-S.

The paper is structured as follows: Section 2 describes the principle of Compositionality. Section 3 discusses the "A - C" paradigm. Section 4 describes the ITL formalization of A - C and its application to Web services. Section 5 discusses OWL-S and SWRL. Section 6 describes how the A - C properties can be expressed as a SWRL representation. Section 5 proposes composition rules and their proof obligations for introducing compositionality in service specifications. Section 8 presents an example of an auction service. Section 9 discusses runtime verification techniques using AnaTempura. Section 10 outlines conclusion and future work.

2. COMPOSITIONALITY

Compositionality [6] refers to the technical property that enables reasoning about a composed system on the basis of its constituent parts without any additional need for information about the implementation of those parts. The notion of compositionality is very important in computer science as it facilitates modular design and maintenance of complex systems. Compositionality is also a desired criterion for verification methodologies, particularly for the development and analysis of large scale systems. The idea was first formulated by Edsger W. Dijkstra [7]. For reasoning satisfactorily about composed system, we only require that systems and their components are specified using *predicates*. The principle of compositionality can be readily applied to Web services. Reasoning about compositions is facilitated using compositional principles, rules and their proof obligations, on the predicates derived from service descriptions stored in some repository by a preprocessor or a reasoning engine.

[1]also referred to as A - C.

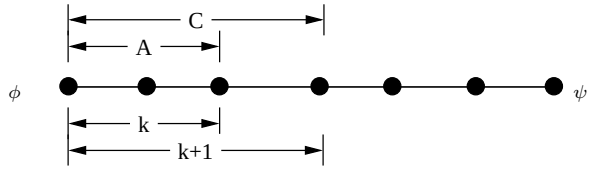

Figure 1: Assumption-Commitment

3. THE ASSUMPTION - COMMITMENT PARADIGM

The A - C framework is a compositional specification methodology. It was first discovered by Jayadev Misra and Mani Chandy [13] as a proof technique for networks of processes executing concurrently via synchronous message passing. A related technique for shared variable concurrency was proposed by Cliff Jones in [11]. The objective is to specify a process within a network. Formally an A - C formula has the following form:

$$(A, C) : \{\phi\}P\{\psi\} \tag{1}$$

where P denotes a process and A, ϕ, ψ, C denote predicates. Informally an A - C formula has the following meaning:

if ϕ holds in the initial state, including the communication history in which P starts its execution then

- *C holds initially and C holds after every communication provided A holds after all preceding communication and*
- *If P terminates and A holds after all previous communication (including the last one) then ψ holds in the final state including the final communication history*

An equivalent definition for (1) using induction (ref. Fig. 1) can be defined as :

1. if ϕ holds initially in P then C holds initially in P.
2. if ϕ holds initially in P and A holds upto the kth point in P, then C holds up to the $k+1$th point for all $k \geq 0$ and,
3. if ϕ holds initially in P and A holds at all times during P, and P terminates, then ψ holds on termination.

Here A expresses an assumption describing the expected behaviour of the environment of P. C expresses a commitment which is guaranteed by the process P as long as the environment does not violate the assumption A and ϕ and ψ express pre- and post-conditions upon the state of P. A and C are required to hold for both terminated and nonterminated computation.

4. FORMALISING ASSUMPTION-COMMITMENT IN ITL

ITL is a flexible notation for both propositional and first-order reasoning about periods of time. Tempura: an executable subset of ITL, provides a framework for developing, analysing and experimenting with suitable ITL specifications. The syntax of ITL is defined in Fig. 2 where μ is an integer value, a is a static variable (doesn't change within an interval), A is a state variable (can change within an interval), v a static or state variable, g is a function symbol and p is a predicate symbol.

ITL contains conventional propositional operators such as $\wedge$, $\neg$ and first order ones such as $\forall$ and =. There are temporal operators like "; *(chop)*", "* *(chopstar)*" and "skip". Additionally in ITL,

Expressions	$e ::=$	$\mu \mid a \mid A \mid g(e_1, \ldots, e_n) \mid \imath a : f$
Formulae	$f ::=$	$p(e_1, \ldots, e_n) \mid \neg f \mid f_1 \wedge f_2 \mid \forall v \bullet f \mid \mathsf{skip} \mid f_1 ; f_2 \mid f^*$

Figure 2: Syntax of ITL

there are temporal operators like $\bigcirc$ and $\Box$. Expressions and Formulae are evaluated relative to the beginning of an interval.

The informal semantics of the most interesting constructs are as follows:

- $ia : f$: the value of a such that f holds.
- skip : unit interval (length 1).
- $f_1; f_2$: holds if the interval can be chopped into a prefix and a suffix interval such that f_1 holds over the prefix and f_2 over the suffix.
- f^* : holds if the interval is decomposable into a number of intervals such that for each of them f holds.

Some of the frequently used abbreviations are listed in Table 1.

Table 1: Frequently used temporal abbreviations

$\bigcirc f$	$\widehat{=}$ skip ; f	next
more	$\widehat{=} \bigcirc$*true*	non-empty interval
empty	$\widehat{=} \neg$*more*	empty interval
$\Diamond f$	$\widehat{=}$ *finite* ; f	sometimes
$\Box f$	$\widehat{=} \neg\Diamond\neg f$	always
fin f	$\widehat{=} \Box(\mathsf{empty} \supset f)$	final state
$\textcircled{w} f$	$\widehat{=} \neg \bigcirc \neg f$	weak next
$\textcircled{\diamond} f$	$\widehat{=} f$; *true*	some initial subinterval
$\boxdot f$	$\widehat{=} \neg(\textcircled{\diamond} \neg f)$	all initial subintervals
$\textcircled{\diamond} f$	$\widehat{=}$ *finite* ; f ; *true*	some subinterval
$\textcircled{a} f$	$\widehat{=} \neg(\textcircled{\diamond} \neg f)$	all subintervals

4.1 Application to Web services

Composed Web services are independently executing components communicating via message passing to yield the desired behaviour. Since the A - C paradigm offers compositional proof techniques for specifying and verifying composed system, communicating via message passing, it lends itself readily for application in the domain of Web services. For our purpose we need to use a variant of the formalism which is somewhat different in spirit from the classical compositional reasoning. We redefine the paradigm as below:

> *The Assumption-Commitment paradigm is a compositional specification and verification technique that can be applied to services or networks of services, composed to execute sequentially or concurrently. The paradigm provides compositional rules whose proof obligations ensures soundness of the composition. The validity of the proof obligations or verification conditions can be checked at the design stage for static analysis of the composed system, and can also be checked at runtime using a theorem prover with actual parameters. The Assumption-Commitment paradigm thus provides a powerful technique for reasoning about services that can be composed and once they are composed, it helps in validating the integrity of the composition at runtime.*

The assertions required for formulating A - C in this scenario are quite general. Conventionally, assumptions are predicates required to be true by the environment of a service. For a service executing as part of a network, the environment is composed of all other services executing in the network. In the original formalism, assumptions are predicates over the channel variables. We relax this notion and require assumption to be an ongoing temporal property including constraints on the input parameters that a service provider demands to be true as long as his service is in execution. The assertions for the commitment can be any temporal property of the service which the provider wishes to expose as a guarantee to the assumption.

4.2 An ITL formalization of Assumption-Commitment

A Service, S in ITL is expressed as a quadruple $(\omega, As, Co, \omega')$ where,

ω	: state formula about initial state
As	: a temporal formula specifying properties about the environment
Co	: a temporal formula specifying properties about the service
ω'	: state formula about final state

Validity of an A-C formula,

$$\models (As, Co) : \{\omega\} S \{\omega'\}$$

inductively defined in ITL, has the following intuitive meaning:

- if ω holds in the initial state, in which S starts its execution then Co holds initially in S.
- if ω holds initially in S and, As holds upto the σ_kth state in S, then Co holds upto the σ_{k+1}th state for all $k \geq 0$.
- if ω holds initially in S and, As holds at all previous states, before S terminates then ω' holds on termination.

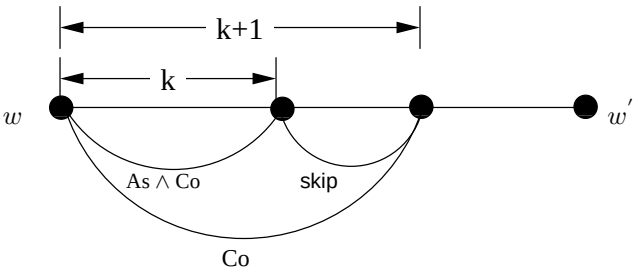

Figure 3: ITL representation of Assumption-Commitment

Formally in ITL, the validity of the A - C representation (ref. Fig. 3) has the following form :

$$(As, Co) : \{\omega\} S \{\omega'\} \stackrel{\text{def}}{=} S \supset \omega \wedge \boxdot((As \wedge Co) ; skip \supset Co) \wedge \textit{fin}\ \omega'$$

5. OWL-S AND SWRL

OWL-S is an OWL ontology for the specification of services. It is structured to provide three types of information about a service. The *service profile* provides a representation of properties and capabilities that can be used by a service requester to specify their needs and service providers to advertise their services for "what" it does. The *process model* describes "how" the service "serves". It provides concepts for specifying the functional attributes of a service in the form of Inputs, (Conditional)Outputs, Preconditions and (Conditional)Effects. It also provides constructs for service composition. The *Grounding* maps process parameters to corresponding WSDL representation of inter-service communication aspects of the service, in terms of the message formats, protocols and communication ports.

Our interest lies in augmenting the Process model with A - C properties. Since assumptions and commitments are temporal formulae, we need an extension to the core ontology that allows us to express them. SWRL [9] is a preliminary proposal for a rule language designed to express rules and constraints in the framework of the OWL language. Related work for specifying rules within the domain of Semantic web are initiatives like RuleML [3] and DRS [8]. However SWRL gives us the flexibility to remain within the domain of OWL. A Concrete XML and RDF syntax are part of the current specification of the language. In this paper we present our example using the RDF/XML encoding. It must be noted that SWRL is evolving and significant changes can be expected in the near future.

SWRL is based on a combination of the OWL DL and OWL Lite sublanguages of the OWL Web Ontology Language with the Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language. It proposes the specification of rules in the form of an implication. The Atoms within the body and the head of the implication can be Class Predicates $C(x)$ or Property Predicates $P(x, y)$. Within the body or head, multiple atoms are treated as a conjunction. Here is a primitive way of how one could express a rule or a condition for a book buying service, "BBS". Semantically, the rule can be defined as,

if a buyer has a valid Account and a valid Credit Card, he can buy a book from BBS.

We have the following variables as inputs to the book buying service,

- Account ID
- Password
- Creditcard number
- Expiry date
- ISBN number

The variables can be defined in SWRL as:

```
<Variable rdf:ID="acctID"/>
<Variable rdf:ID="password"/>
<Variable rdf:ID="creditCardNumber"/>
<Variable rdf:ID="expDate"/>
<Variable rdf:ID="ISBNNumber"/>
```

The rule can now be stated as:

```
<Imp>
 <body rdf:parseType="Collection">
  <individualPropertyAtom>
   <propertyPredicate rdf:resource="#accountExists"/>
    <argument1 rdf:resource="#acctID" />
    <argument2 rdf:resource="#password" />
  <individualPropertyAtom>
  <individualPropertyAtom>
   <propertyPredicate rdf:resource="#creditExists"/>
    <argument1 rdf:resource="#creditCardNumber" />
    <argument2 rdf:resource="#expdate" />
  <individualPropertyAtom>
 <body>
 <head rdf:parseType="Collection">
  <individualPropertyAtom>
   <propertyPredicate rdf:resource="#allowedToBuy"/>
    <argument1 rdf:resource="#acctID"/>
    <argument2 rdf:resource="#ISBNNumber"/>
   <individualPropertyAtom>
 <head>
<Imp>
```

Many times it may be the case that we do not want to express an implication, but simply a predicate or a conjunction of predicates [15]. This is the case for expressing "Condition"- which is used in many places within the Process model. Within the framework of OWL-S, Conditions are required to be logical formulae. They are part of the definition for "Precondition" - a kind of Condition and also for Conditional outputs and effects. Currently SWRL does not specify any such constructs. Several proposals have been discussed for expressing Conditions in this format [15]. The common factor in all these proposals is having a top-level concept called "`Formula`" and defining "`Condition`" to be of rdf:type `Formula`. Eventually, "`Formula`" is expected to be defined as a part of SWRL. However for practical purposes it can be defined within the Process model or DRS representation of the same can be used. In this paper we abstract the top -level representation of "`Formula`", as we believe that this is a matter of namespace representation and express temporal properties as Property Predicates using SWRL.

6. A SWRL REPRESENTATION OF A - C PROPERTIES

A - C properties are Temporal formulae[2].

```
<owl:Class rdf:ID="TemporalFormula">
  <rdf:subClassOf rdf:resource="#Formula">
</owl:Class>

<owl:Class rdf:ID="Assumption">
  <rdf:type rdf:resource="#TemporalFormula">
</owl:Class>

<owl:Class rdf:ID="Commitment">
  <rdf:type rdf:resource="#TemporalFormula">
</owl:Class>
```

From the aspect of ontological representation, A - C can be regarded as first-order logical formulae analogous to Conditions and augmented with temporal operators. We have modelled a basic ontology for expressing various temporal operators in ITL [1]. When the representation of Conditions is standardized, assumption and commitments can be easily expressed by using the two ontologies. As an example consider the following temporal formula:

$$\Box validISBNNumber(BookTitle, ISBN)$$

Informally it means that *always* the name of the book should correspond to the given ISBN number. The ontological representation of the above is,

```
<individualPropertyAtom>
  <prefixOperator rdf:resource="&itl;Always"/>
    <propertyPredicate rdf:resource=
                "#validISBNNumber"/>
    <argument1 rdf:resource="#bookTitle"/>
```

[2]We do not specify namespaces of `Formula` and `TemporalFormula` at this stage.

```
    <argument2 rdf:resource="#ISBN"/>
</individualPropertyAtom>
```

where, `prefixOperator` is a Class, representing temporal operators like $\Box, \boxdot, \boxplus, \Diamond$ and $\neg$. The $\bigcirc$ operator can be applied to both formulae and variables.

7. COMPOSITIONAL RULES FOR SERVICE COMPOSITION

In this section, we explain how A - C can be used to compositionally reason about integrated services. Web services compositions can be realised using several programming language control constructs. Predominantly there are two forms of compositions: sequential and parallel. Other forms of compositions can be derived from these two forms using constructs like *if-then-else, iterate, repeat-until, while-do and choice* [14].

We define compositional rules using A - C for the most intuitive form of composition i.e sequential and parallel composition. We provide the ontological representation for pre/post conditions and A - C predicates using SWRL. In following subsections:

1. ω, $\omega^{'}$ represent pre/post conditions respectively.
2. As_1, As_2 represent assumption for service S_1 and S_2.
3. Co_1, Co_2 represent commitment for service S_1 and S_2.
4. As, Co represent assumption and commitment of the composition.

7.1 Sequential Composition

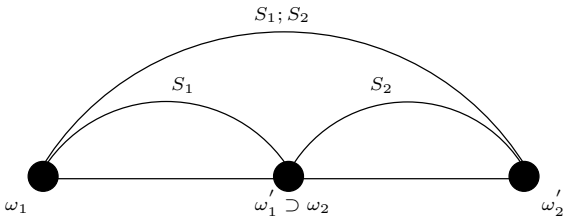

Figure 4: Sequential Composition

We consider the sequential composition (ref. Fig. 4) of two services. Services S_1 and S_2. In case of sequential composition we

$$
\begin{array}{rcll}
(As, Co) & : & \{\omega_1\} S_1 \{\omega_1^{'}\} & (1) \\
(As, Co) & : & \{\omega_2\} S_2 \{\omega_2^{'}\} & (2) \\
As & \equiv & \boxplus As & (3) \\
Co & \equiv & Co^{*} & (4) \\
\omega_1^{'} & \supset & \omega_2 & (5) \\
\hline
(As, Co) & : & \{\omega_1\} S_1; S_2 \{\omega_2^{'}\} & (6)
\end{array}
$$

require that As and Co are respective fixpoints[3] of the ITL operators $\boxplus$ and $chopstar$. For the commitment to hold over the interval defined by $S_1; S_2$, we first require the assumption to hold over that interval. Hence, if the assumption holds and if it is a fixpoint of $\boxplus$, it guarantees to holds over the individual subintervals i.e intervals defined by S_1 and S_2 as well (semantics of $\boxplus$). Now in response to this assumption, if the services guarantee some commitments, they hold on the individual subintervals for S_1 and S_2. If we choose these commitments such that they are fixpoints of $chopstar$(i.e a singular commitment), we can easily collapse the commitment to hold for the interval defined by $S_1; S_2$. The advantage of these restrictions are ease in implementation and reduced complexity in validating the composition.

We take an example of a global book searching service composed in sequence with a book buying and shipping service. The composition engine requires all necessary user inputs, i.e ISBN number of the book, credit card details and shipping details to be supplied to the composite service before engaging into the composition. Pre/post conditions can be defined as:

$$
\begin{aligned}
\omega &\ \widehat{=}\ valid(ISBN) \wedge validCreditCard(cardNumber) \\
\omega^{'} &\ \widehat{=}\ cardBilled(cardNumber) \wedge \\
&\quad\ bookShipped(shippingAddress)
\end{aligned}
$$

When expressed as a property predicate in SWRL, the Precondition: can be defined as:

```
<conjuncts rdf:parseType="Collection">
  <individualPropertyAtom>
    <propertyPredicate
        rdf:resource="#validISBNNumber"/>
      <argument1 rdf:resource="#bookTitle"/>
      <argument2 rdf:resource="#ISBN"/>
  </individualPropertyAtom>
  <individualPropertyAtom>
    <propertyPredicate
        rdf:resource="#validCreditCard"/>
      <argument1 rdf:resource="#cardNumber"/>
      <argument2 rdf:resource="#expDate"/>
  </individualPropertyAtom>
</conjuncts>
```

The post-condition can be expressed accordingly.

However, it can be observed that these assertions do not make any statements about the credit card validity by the requester throughout the composition and the assurance that the card will not be billed till the transaction is complete by the service provider.
A - C assertions are required to increase trustworthiness of the service and to take corrective measures in case of any unexpected behaviour. For e.g., it is likely that the requester's credit card ceases to be valid during an ongoing transaction. Therefore the following temporal assertions are required to be made part of the specification.

$$
\begin{aligned}
As &\ \widehat{=}\ \boxplus validCreditCard(cardNumber) \\
Co &\ \widehat{=}\ (\neg cardBilled(cardNumber))^{*}
\end{aligned}
$$

The commitment can be expressed as:

```
<individualPropertyAtom>
  <suffixOperator rdf:resource="&itl;Chopstar"/>
  <prefixOperator rdf:resource="&itl;Negate"/>
    <propertyPredicate rdf:resource="#cardBilled"/>
    <argument1 rdf:resource="#amonut"/>
    <argument2 rdf:resource="#cardNumber"/>
</individualPropertyAtom>
```

The assumption can be expressed accordingly.

7.2 Parallel Composition

A network of services executing concurrently consists of a set of services and a set of shared objects such as channels, through which the services communicate via message passing. The specification, proof obligations and the compositional rule for services executing in parallel are as defined below: For services executing concurrently, the environment of each service is governed by the environment of every other service in the network and by the overall environment of the composition. Hence the proof obligation

[3]The fixed point of a function, f is any value, x for which $f\ x = x$. A function may have any number of fixed points from none (e.g. $f\ x = x+1$) to infinitely many (e.g. $f\ x = x$).

$$
\begin{array}{rcll}
(As_1, Co_1) & : & \{\omega_1\} S_1 \{\omega_1^{'}\} & (1) \\
(As_2, Co_2) & : & \{\omega_2\} S_2 \{\omega_2^{'}\} & (2) \\
As \wedge Co_1 & \supset & As_2 & (3) \\
As \wedge Co_2 & \supset & As_1 & (4) \\
\hline
(As, Co_1 \wedge Co_2) & : & (\omega_1 \wedge \omega_2) S_1 \parallel S_2 (\omega_1^{'} \wedge \omega_2^{'}) & (5)
\end{array}
$$

for parallel composition, relates the environment (As_1) of a service ((S_1) with commitment (Co_2) of the other service, (part of the environment of (S_1)) as the observable influence and with the assumption (As) of the overall composition. An example of parallel composition is presented in the following section.

8. CASE STUDY: AN AUCTION SERVICE

An Auction service (ref. Fig. 5) is presented as an example of a composite service. The Auction service is a composition between several buying services, a selling service and the auction house, executing in parallel with each other. Buyers and seller are classified

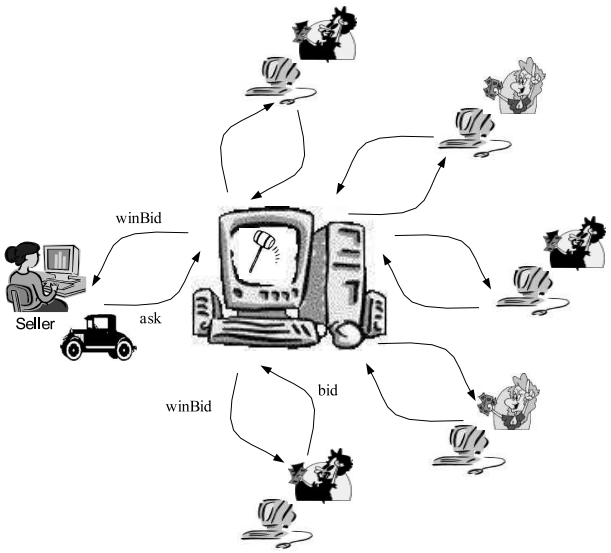

Figure 5: Composition of an Auction Service

as bidders in the auction process. Seller submits an *ask* price to the auction house, buyers submit *bids* as per the rules of the auction, the auction house validates the incoming bids and clears the auction, declaring the winning bid for each clearance. Finally the winner is announced and the auction is closed.

The auction process spans over an interval with intermediate states being defined at the instants where communication between the partners take place. The execution of the composed service takes place concurrently as buyers can submit bids while other bids are being processed by the auction house. The Seller can also change his ask price while the auction progresses, depending on the market situation. However for simplicity we do not consider that case here and model the composition only between the Auction house and the buying services.

In our design we assume the selling and buying services to be thin clients of the auction house. The auction house itself is a thick computational server. It validates rules of the auction and properties of the incoming bids. Some of these properties can be defined informally as below:

- *Registration*: To trade via the Auction house bidders (seller/buyers) have to register with the Auction house as members.
- *Beat-the-Quote*: At any time, the buyer cannot lower the current highest amount.
- *Unique Bid*: At any time, a buyer has only one active bid.
- *Winning Bid*: At any time, only the winning bid is sent back to the buyer.
- *Dominant Bid*: At any time, the latest bid submitted by a buyer has to be higher than the last bid submitted by him.
- *Unique Winner*: The auction house guarantees a unique winner once the auction is over.

Registration for the auction is a *precondition* for every buying/selling service. The *postcondition* is that the auction declares a unique winner. The remaining rules are constraints on bids and are required to hold while the auction is in progress. They can be considered as the "assumption" of the auction house or "commitments" by the buying service. As long as these assumptions are satisfied, the auction house guarantees to admit the bidder for the next round and consider the bid as a valid bid.

8.1 Formalization of the Auction Service

The auction spans over an interval (ref. Fig. 6) defined by a sequence of n states. The ask price is submitted at state 0 and the winning bid is declared at state n. Bids are submitted and the auction is cleared at all intermediate states i.e between states $1 \ldots n-1$. The number of buyers registered for auction is k. Communication between the buying services and the auction house takes place via channels.

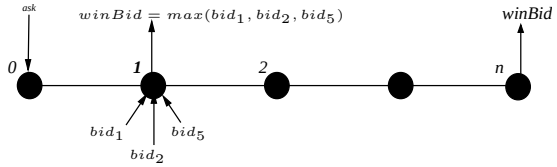

Figure 6: Observable States of the Auction House

We define observable variables for the auction service in Table 2 below.

Table 2: Observable variables

Ask price	$\widehat{=} ask$
Bidder, i's bid	$\widehat{=} bid_i : 0 < i \leq k$
list of bids at any state	$\widehat{=} bidList$
	$\widehat{=} \{bid_i \mid 0 < i \leq k\}$
Winning Bid at any state	$\widehat{=} max(bidList)$
	$\widehat{=} winBid$
Bid received by a buyer	$\widehat{=} bidrec$

8.2 Specifying the Auction House

The initial-final state properties required to be validated by the auction house can now be defined as,

$$
\begin{array}{rcl}
\omega & \widehat{=} & winBid = ask \wedge \\
 & & \forall i : 0 < i \leq k : isRegistered(i) \\
\omega^{'} & \widehat{=} & \exists i : 0 < i \leq k : isWinner(i)
\end{array}
$$

where, $isRegistered(i)$ and $isWinner(i)$ are pre/postcondition predicates, respectively. The assumption (As_{ah}) for any bid submitted to the auction house by a bidder and the corresponding commitment by the auction house Co_{ah} can be formally expressed as,

$\forall i : 0 < i \leq k$

$$As_{ah} \;\widehat{=}\; \Box((\bigcirc bid_i \geq bid_i) \wedge (\bigcirc bid_i \geq winBid))$$
$$Co_{ah} \;\widehat{=}\; \Box(bidrec = winBid)$$

We express the Assumption predicate in SWRL as below:

```
<Variable rdf:ID="bid"/>
<Variable rdf:ID="winBid"/>
<Variable rdf:ID="bidrec"/>

<conjuncts rdf:parseType="Collection">
  <individualPropertyAtom>
    <prefixOperator rdf:resource="&itl;Always"/>
    <propertyPredicate rdf:resource="#isGEQNext"/>
      <argument1 rdf:resource="#bid" >
        <prefixOperator rdf:resource="&itl;Next"/>
      </argument1>
      <argument2 rdf:resource="#bid" />
  </individualPropertyAtom>
  <individualPropertyAtom>
    <prefixOperator rdf:resource="&itl;Always"/>
    <propertyPredicate rdf:resource="#isGEQWinBid"/>
      <argument1 rdf:resource="#bid" />
        <prefixOperator rdf:resource="&itl;Next"/>
      <argument2 rdf:resource="#winBid" />
  </individualPropertyAtom>
</conjuncts>
```

Applying the A-C formalism for these compositional properties,

$$\boxed{i}((As_{ah} \wedge Co_{ah})\,;skip \supset Co_{ah})$$

to the auction house service specification, we have the following compositional ITL formula that is required to be validated while the auction is in progress.

$$\boxed{i}(\Box((\bigcirc bid_i \geq bid_i) \wedge (\bigcirc bid_i \geq winBid) \wedge (bidrec = winBid))\,;skip \supset \Box(bidrec = winBid))$$

8.3 Specifying the Buying Service

We focus on the A-C properties of the buying service. These are informally described below. The compositional property of assumption for any bid received by the buying service, from the auction house, and the corresponding commitment by the buying service, can be formally expressed as, $\forall i : 0 < i \leq k$

$$As_i \;\widehat{=}\; \Box(bidrec = winBid)$$
$$Co_i \;\widehat{=}\; \Box((\bigcirc bid_i \geq bid_i) \wedge (\bigcirc bid_i \geq bidrec))$$

We express the Assumption predicate in SWRL as below:

```
<Variable rdf:ID="bid"/>
<Variable rdf:ID="winBid"/>
<Variable rdf:ID="bidrec"/>

<individualPropertyAtom>
  <prefixOperator rdf:resource="&itl;Always"/>
  <propertyPredicate rdf:resource="#isEQ"/>
    <argument1 rdf:resource="#bidrec" />
    <argument2 rdf:resource="#winBid" />
</individualPropertyAtom>
```

Analogous to the auction house service we apply the A-C formalism,

$$\boxed{i}(\Box((bidrec = winBid) \wedge (\bigcirc bid_i \geq bid_i) \wedge (\bigcirc bid_i \geq bidrec))\,;skip \supset \Box((\bigcirc bid_i \geq bid_i) \wedge (\bigcirc bid_i \geq bidrec)))$$

8.4 Composing the Auction Service

The auction house, buying service and selling service execute concurrently. We simplify the scenario by considering composition only between the buying service and the auction house. The environment of the overall composition, i.e As does not impose any constraints on the composition and defaults to true. The proof obligations for services composed in parallel are recalled from Section 7.2.

$$Co_2 \supset As_1, Co_1 \supset As_2$$

The proof obligations for the auction service can now be specified as, $\forall i : 0 < i \leq k$,

$$Co_{ah} \supset As_i$$
$$Co_i \supset As_{ah}$$

The validity of above proof obligations can be proved from Section 8.2 and 8.3.

9. RUNTIME VERIFICATION USING ANATEMPURA

An important area where compositional specifications could provide valuable assistance is verification of service composition. Traditional methods of verifying a composed system are noncompositional and require a global examination of the entire system. Approaches such as model checking also fail to scale up well since the global state space that has to be explored grows exponentially as components are added to the system. However, application of these methods to verify the composition of web services is problematic because the actual binding between various components is dynamic and therefore there is no global system available a priori for applying these verification techniques. Compositional verification, however, shifts the burden of verification from the global level to the local component level. Hence global properties of the composition can be established by composing together independently verified component properties.

The assumption-commitment paradigm, a deductive (proof - theoretic) technique can be effectively applied as a compositional verification technique for Web services composition. The idea is to prove the validity of the proof obligation for the composition using the assertional specification of each service in the network. The verification can be undertaken at two stages during service composition: (a) At the design stage where decisions regarding which services can be composed are to be made. Here the verification can be automated using a theorem prover;(b) At runtime, when services are actually wired with each other at the ports.

We are more interested in verifications of the second kind. The motivation for that stems from the fact that at runtime service composition can be influenced by several factors like network conditions, synchronization and availability of individual services in the network. Dynamic coordination can thus give rise to an emergent behaviour which may not be desired. The purpose of verification at runtime is therefore to gauge such unwanted behaviour, that may lead to a "chaotic" composition. Since the assertions we propose are temporal properties of services and their environment, the proof obligations for the specification of the composition have to be validated by an engine capable of handling temporal properties AnaTempura [2] (ref. Fig 7) is a tool for the runtime verification of systems. It is an interpreter for executable Interval Temporal Logic specifications written in Tempura: a subset of ITL. AnaTempura generates a state-by-state analysis of the system behaviour as the computation progresses. At any state during the computation if the

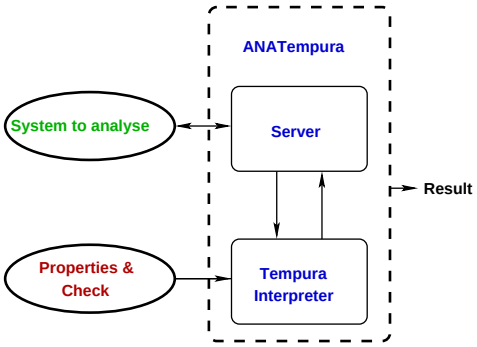

Figure 7: Runtime verification using Anatempura

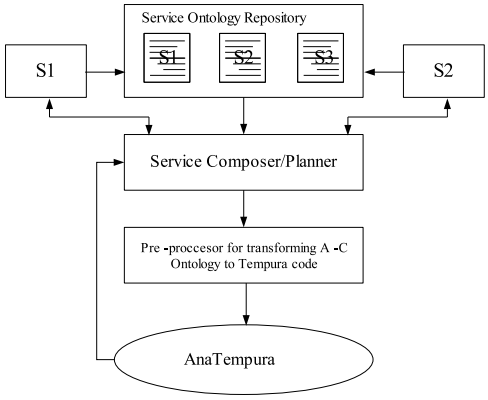

Figure 8: Framework for service composition using Anatempura

variable values cease to satisfy the Tempura formula, AnaTempura throws up an exception for that state.

For verification, the proof obligations that encode the temporal assertions are specified in Tempura. At runtime, the assertions are validated, by passing the actual parameter values to the Tempura Program (ref. Fig. 8) at the initial state and at each critical state defined by the service provider. AnaTempura validates the proof obligations at these states. If the proof obligations cease to hold, it implies that some form of unwanted or chaotic behaviour has occurred. This kind of verification serves two purposes: (a) It assists in identifying the errors in service description as the specification emerges from there. Conventional ways of verifying compositions work at the implementation level using techniques like exception handling;(b) Third party arbitration services can use the mechanism for monitoring quality-of-service parameters. This is because the verification mechanism still works at the interface level and therefore no implementation details are required. We have developed a preliminary implementation of the auction example using this technique which is currently in the testing phase.

10. CONCLUSION AND FUTURE WORK

In this paper we provide the much needed theoretical background for applying compositionality to the domain of Semantic Web services. We believe that both specification and verification should be highly compositional allowing modular validation and verification to be performed. We have shown how Semantic Web service specifications written as OWL ontologies, can be reinforced with temporal properties - Assumption and Commitment. We have chosen OWL-S as it has an inbuilt provision for accommodating logical formulae, which is missing in other languages. We have argued that apart from state predicates ongoing assertions are also required to fully capture the behaviour of a service, specified as a black box. We have shown how assumption and commitment can be specified for compositional reasoning about semantic Web service composition, using OWL-S and SWRL. We also show how the theory can be applied in practice to the composition of an auction service. We have done a preliminary implementation of the auction service which will be presented in a future paper.

Little work has been done in the area of compositional specification and verification of services as revealed from the literature review done so far in the domain of semantic web services, both in academia and industry. Semantics for the process model has been defined by Narayan and McIlraith [20] using axioms in situation calculus, which are then mapped to Petri-net representation. An alternative Concurrent Execution semantics for the same has also been proposed by Ankolekar et al [4]. However, these do not discuss compositionality and verification techniques that are imperative while composing services on-the-fly. Several industrial efforts to create service composition standards like BPEL4WS [24] and WSCI [10] provide syntactical means of describing and composing services. They however, lack the formal framework needed for verification of services composed using such specifications.

As part of our future goal, we aim to build a pre-processor (ref. Fig. 8) that converts assumption - commitment specification within an OWL-S ontology into executable Tempura specification. The tool would be part of a larger framework for service composition. We have already designed the framework for the implementation of such a tool. We also plan to extend our work on verification of services using AnaTempura.

11. REFERENCES

[1] An Ontology for ITL. *http://www.cse.dmu.ac.uk/~monika/Pages/Ontologies/ITLOnto.owl*.

[2] ITL and (Ana)Tempura Home page on the web. *http://www.cse.dmu.ac.uk/~cau/itlhomepage/itlhomepage.html*.

[3] The Rule Markup Initiative. http://www.dfki.uni-kl.de/ruleml/.

[4] Anupriya Ankolekar, Frank Huch, Katia Sycara. Concurrent Execution Semantics for DAML-S with Subtypes. The First International Semantic Web Conference (ISWC), Sardinia (Italy), June 2002.

[5] D. Fensel, C. Bussler. Web Services Modelling Framework, 2002. http://www.swsi.org/resources/wsmf-paper.pdf.

[6] W.-P. de Roever et al. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*. Cambridge University Press, Cambridge, England, 2001.

[7] E. W. Dijkstra. Solution of a problem in concurrent programming control. Number 8(9). CACM, 1965.

[8] Drew McDermott and Dejing Dou . Representing Disjunction and Quantifiers in RDF Embedding Logic in DAML/RDF. International Semantic Web Conference, 2002.

[9] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML . Technical report, Version 0.5 of 19 November 2003.

[10] Intalio, Sun Microsystems, BEA Systems, SAP. Web Service Choreography Interface (WSCI) 1.0 Specification, 2002.

[11] Jones C.B. Specification and design of (parallel) programs. pages 321–332, Amsterdam, 1983. Proceedings of information processing'83, North Holland Publishing Co.

[12] M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. Guinness, P. F. Patel-Schneider, L. A. Stein.

Web Ontology Language (OWL) W3C Reference version 1.0, 18 August 2003. *http://www.w3.org/TR/2002/WD-owl-ref-20021112*.

[13] J. Misra and K. Chandy. Proofs of networks of processes. volume 7(7):417-426. IEEE Transactions on Software Engineering, 1981.

[14] Monika Solanki, Antonio Cau, Hussein Zedan. Introducing compositionality in webservice descriptions. Paris, France, 2003. 3rd International Anwire Workshop on Adaptable Service Provision, Springer-Verlag.

[15] Monika Solanki, Drew McDermott and David Martin. Discussions on the Semantic Web service Interest Group mailing list. *http://www.w3.org/2002/ws/swsig/*.

[16] B. Moszkowski. *Executing temporal Logic Programs.* Cambridge University Press, Cambridge, England, 1986.

[17] B. Moszkowski. *Programming Concepts, Methods and Calculi, IFIP Transactions, A-56.*, chapter Some Very Compositional Temporal Properties, pages 307–326. Elsevier Science B. V., North-Holland, 1994.

[18] B. Moszkowski. *Compositionality: The Significant Difference*, volume 1536 of LNCS, chapter Compositional reasoning using Interval Temporal Logic and Tempura, pages 439–464. Springer Verlag, Berlin, 1996.

[19] Motta, E., Domingue, J., Cabral, L. and Gaspari, M. IRS-II: A Framework and Infrastructure for Semantic Web Services. In *Proceedings of the 2nd International Semantic Web Conference*, 2003.

[20] S. Narayan and S. A.Mcllraith. Simulation, Verification and Automated Composition of Web Services. Hawaii, USA, 2002. Eleventh International World Wide Web Conference.

[21] Roberto Chinnic, Martin Gudgin,Jean-Jacques Moreau, Sanjiva Weerawarana. Web Services Description Language (WSDL) Version 1.2, 2003. http://www.w3.org/TR/2003/WD-wsdl12-20030124/#intro.

[22] M. Sheshagiri, M. desJardins, and T. Finin. A Planner for Composing Services Described in DAML-S. In *Proceedings of the AAMAS Workshop on Web Services and Agent-based Engineering*, 2003.

[23] The OWL-S Coalition. OWL-S 1.0 (Beta) Draft Release., 2003. *http://www.daml.org/services/owl-s/1.0/* .

[24] Tony Andrews et al. Business Process Execution Language for Web Services, Version 1.1, 2003. http://www-106.ibm.com/developerworks/library/ws-bpel/.

[25] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S Services Composition Using SHOP2. In *Proceedings of the 2nd International Semantic Web Conference*, 2003.