Introducing Compositionality in Web service Descriptions

Monika Solanki, Antonio Cau, Hussein Zedan
Software Technology Research Laboratory,
De Montfort University,
The Gateway, Leicester LE1 9BH, UK
monika, acau, zedan@dmu.ac.uk

Abstract

Web services are essentially black box components from a composer's or a mediator's perspective. The behavioural description of any service can be asserted by the composer, only through interface predicates exposed by the service provider. Normally for proving properties of service compositions, 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 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.

1 Introduction and Motivation

Current Web service description languages suffer from the lack of their ability to provide constructs and concepts that enable reasoning about service behaviour during execution. Languages like WSDL[15] and BPEL4WS[2] 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[18], an ontology for describing Web services in the domain of the Semantic web[17], does provides concepts like preconditions and conditional effects. The WSMF[5] defines pre/post conditions for services. However pre/post-conditions and effects are required to hold only at the initial/final states of the service execution.

When the service requester (human/software) has no control over the various stages of service execution, it becomes necessary to enrich the service description with certain properties which enable dynamic reasoning about service behaviour. The need for such rich specifications arises while reasoning about the composition and verification of services. Current composition planners/engines [19, 16] use input/output mapping, type characteristics of these parameters and pre/post conditions to generate compositions/plans. However to ensure a sound composition, services need to be composed using a specification technique that characterizes runtime behaviour of the service. Composers have no means to validate an ongoing composition and take appropriate measures in case the specifications cease to be satisfied. Hence behavioural specification of services should also include assertions that characterise intermediate critical decision making states during computation. The rationale becomes more apparent while composing services that execute concurrently, due to possibilities of synchronization and communication. Runtime verification is reduced to checking the assertions using an engine, designed to handle temporal properties.

We propose a methodology to augment the specification of a service with temporal properties, formally called *assumption* and *commitment*[1]. The properties are specified in Interval Temporal Logic(ITL)[12, 13, 11, 1], 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. Further, we show that such assertional specifications are compositional and this strategy can be effectively applied for the verification of a composed service. The formalism thus provides a powerful technique for reasoning about service composition, execution and runtime verification of service behaviour. Our approach differs from conventional approaches as we consider validation and verification to be an

[1] Henceforth referred to as A - C

integral part of service composition. This 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 proposes composition rules and their proof obligations for introducing compositionality in service specifications. Section 6 presents a case study. Section 7 discusses runtime verification techniques using AnaTempura[1]. Section 8 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 information about the implementation of those parts. The purpose of a compositional verification approach is to shift the burden of verification from the global level to the local component level. 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 predicates derived from service descriptions stored in some repository by a reasoning engine. Compositional specification also assist in asserting runtime behaviour of the composed system.

3 The Assumption - Commitment Paradigm

The A - C framework is a compositional specification methodology. It was first discovered by Jayadev Misra and Mani Chandy[8] 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 [4]. The objective is to specify a process within a network. Formally an A - C formula has the following form:

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

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*

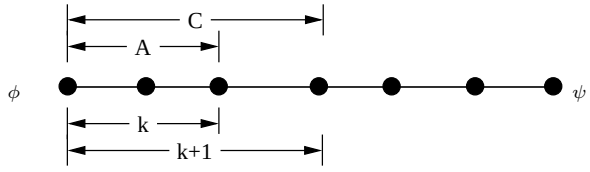

Figure 1. Assumption-Commitment

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, 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.

Expressions	$e ::=$	$\mu \mid a \mid A \mid g(e_1, \ldots, e_n) \mid \imath a \colon f$
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

- 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
$\Diamond\!\!\!\!\cdot\, f$	$\widehat{=} f$; *true*	some initial subinterval
$\boxdot f$	$\widehat{=} \neg(\Diamond\!\!\!\!\cdot\, \neg f)$	all initial subintervals
$\Diamond\!\!\!\!\cdot^{\mathrm{a}}\, f$	$\widehat{=}$ *finite* ; f ; *true*	some subinterval
$\boxdot^{\mathrm{a}} f$	$\widehat{=} \neg(\Diamond\!\!\!\!\cdot^{\mathrm{a}}\, \neg f)$	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
$\omega^{'}$: 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 $\omega^{'}$ holds on termination.

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

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

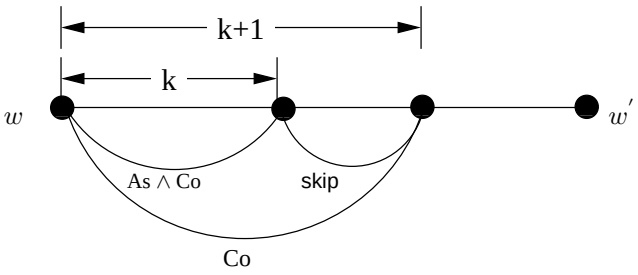

Figure 3. *ITL representation of Assumption-Commitment*

5 Compositional Rules for Service Composition

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*[9]. We define compositional rules using A - C for the two most intuitive forms of compositions. 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.

5.1 Sequential Composition

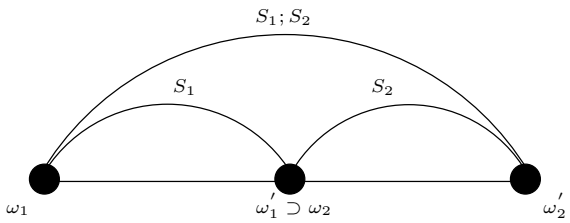

Figure 4. Sequential Composition

We consider the sequential composition (ref. Fig. 4) of two services. Services S_1 and S_2. In case of sequential composition we require that As and Co are respective fixpoints[2]of the ITL operators $\boxed{a}$ 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 $\boxed{a}$, it guarantees to holds over the individual subintervals i.e intervals defined by S_1 and S_2 as well (semantics of $\boxed{a}$). 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.

[2]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).

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

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:

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

However as we see from the above, 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. Therefore the following temporal assertions are required to be made part of the specification.

$$
\begin{array}{rcl}
As & \widehat{=} & \boxed{a}\, validCreditCard(cardNumber) \\
Co & \widehat{=} & (\neg cardBilled(cardNumber))^{*}
\end{array}
$$

5.2 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 for parallel composition, relates the environment (As_1) of a

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

service ((S1) 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.

6 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 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.

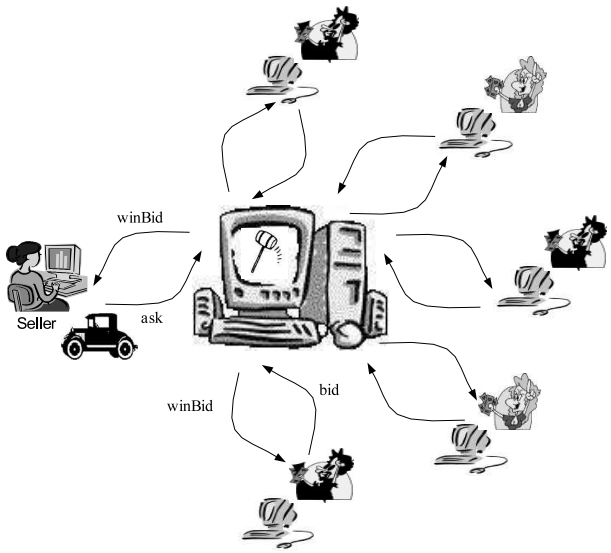

Figure 5. Composition of an Auction Service

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.

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

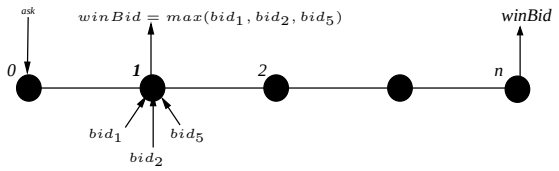

Figure 6. Observable States of the Auction House

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	$\widehat{=} bidrec$

6.2 Specifying the Auction House

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

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

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

$$\begin{aligned} As_{ah} &\widehat{=} (\bigcirc bid_i \geq bid_i) \wedge (\bigcirc bid_i \geq winBid) \\ Co_{ah} &\widehat{=} bidrec = winBid \end{aligned}$$

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.

$$\begin{aligned} &\boxed{i}(((\bigcirc bid_i \geq bid_i) \wedge (\bigcirc bid_i \geq winBid) \wedge \\ &\quad (bidrec = winBid))\,;\, skip \supset (bidrec = winBid)) \end{aligned}$$

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

$$\begin{aligned} As_i &\widehat{=} bidrec = winBid \\ Co_i &\widehat{=} (\bigcirc bid_i \geq bid_i) \wedge (\bigcirc bid_i \geq bidrec) \end{aligned}$$

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

$$\begin{aligned} &\boxed{i}(((bidrec = winBid) \wedge (\bigcirc bid_i \geq bid_i) \wedge \\ &\quad (\bigcirc bid_i \geq bidrec))\,;\, skip \supset (\bigcirc bid_i \geq bid_i) \wedge \\ &\qquad\qquad (\bigcirc bid_i \geq bidrec)) \end{aligned}$$

6.4 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 5.2.

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

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

$$\begin{aligned} Co_{ah} &\supset As_i \\ Co_i &\supset As_{ah} \end{aligned}$$

The validity of above proof obligations can be proved from section 6.2 and 6.3.

7 Runtime Verification using AnaTempura

Verification of Web services can be done at two stages: (a)At the design stage using a theorem prover and (b)At runtime. Service composition can be influenced by several factors at runtime, 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. Since the assertions we propose are temporal properties of services and their environment, the proof obligations for the specification of the composition need to be validated by an engine capable of handling temporal properties.

AnaTempura[1] 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 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 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 are still investigating this technique for practical applicability and this is a work-in-progress.

8 Conclusion and Future Work

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 Web services both in academia and industry. Semantics for the process model has been defined by Narayan et al. [14] using axioms in situation calculus, which are then mapped to Petrinet representation. An alternative Concurrent Execution semantics for the same has also been proposed by Ankolekar et. al [3]. 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 [2] and WSCI [7] provide syntactical means of describing and composing services. They however lack the formal framework needed for verification of services composed using such specifications.

In this paper we provide the much needed theoretical background for applying compositionality to the domain of Web services. We believe that both specification and verification should be highly compositional allowing modular validation and verification to be performed. The proposed framework can be readily applied for execution monitoring in exisiting process models as provided by languages like OWL-S. We have shown how the principles of compositionality and specifically the assumption-commitment paradigm can be readily applied in order to reason about service composition and verification, using not only initial and final state predicates but also critical intermediate states. To support the theory we have proposed rules and their corresponding proof obligations for the most intuitive programming constructs. We then 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. We plan to extend our work on verification of services further and develop a practical approach using AnaTempura.

References

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

[2] Francisco Curbera, Yaron Goland, Johannes Klein, Frank Leymann, Dieter Roller, Satish Thatte, Sanjiva Weerawarana. *Business Process Execution Language for Web Services, Version 1.0*, 2002. *http://www-106.ibm.com/developerworks/library/ws-bpel/*.

[3] 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.

[4] J. CB. Specification and design of (parallel) programs. pages 321–332, Amsterdam, 1983. Proceedings of information processing'83, North Holland Publishing Co.

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

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

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

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

[11] 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.

[12] 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.

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

[14] 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*.

[15] 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.

[16] T. Berners-Lee, J.Hendler and Ora Lassila. *The Semantic Web*. 2000.

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

[18] 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.