Introducing Compositionality in Webservice Descriptions

Monika Solanki, Antonio Cau, Hussein Zedan

Software Technology Research Laboratory,
De Montfort University,
The Gateway, Leicester LE1 9BH, UK
E-mail {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 therefore be asserted by the composer, only through interface predicates exposed by the service provider through the service description. Normally for proving properties of service compositions, pre and post conditions are found to be sufficient and are specified for services described in languages like OWL-S. However these properties, are assertions only on the initial and final states of the service respectively. They do not help in specifying or verifying ongoing behaviour of an individual service or a composed system. In an environment, where services are black boxes, and composition is undertaken at runtime, it is necessary to have additional properties that can be asserted at various points during service execution to ensure that the composition would eventually result in one of the outcomes specified by the service provider. We propose a framework for enriching service descriptions with two additional assertions: assumption and commitment that facilitate compositional reasoning about services for the 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 webservice 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[14], 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- 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 in the sense that 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 requestor (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. Service descriptions annotated with temporal assertions over periods of time, can be viewed as an augmentation or generalization of the use of pre- and post-conditions which are limited to assertions over single states. The need for such rich specifications becomes evident while reasoning about the composition of services and verification of the composition. Current composition planners/engines [18, 16] 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. Because 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. Hence behavioural specification of services should also 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, while in execution. 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. Further 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 specification of a service, regardless of the language used for description, with temporal properties, formally called *assumption* and *commitment*. The properties are specified in Interval Temporal Logic(ITL)[11, 12, 10, 1], our underlying formalism for reasoning about service behaviour over periods of time rather than just the initial and final states. 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 *compositional 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 and validation of service behaviour.

The paper is structured as follows: section 2 describes the principle of Compositionality. Section 3 discusses "Assumption-Commitment" - a compositional specification and proof technique. Section 4 describes the ITL based treatment of Assumption-Commitment and its application to Webservices. Section 5 proposes composition rules and their proof obligations for introducing compositionality in

service specifications. Section 6 discusses runtime verification techniques using an inhouse tool: AnaTempura. Section 7 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 constituents parts without any additional need for 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, so that global properties are established by composing together independently (specified) and verified properties. 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 Webservices. Reasoning about compositions is facilitated using compositional principles, rules and their proof obligations, on the predicates, derived from their descriptions stored in some repository, by a preprocessor or a reasoning engine. Compositional specification also assist in asserting runtime behaviour of the composed system. The compositional specification for webservices should include as part of its observables, assertions on initial, final and critical intermediate states of execution.

3 The Assumption - Commitment Paradigm

The assumption-commitment (A-C) framework is a compositional specification methodology. It was first discovered by Jayadev Misra and Mani Chandy[9] as a proof technique for networks of processes executing concurrently via synchronous message passing. A related technique for shared variable concurrency was done by Cliff Jones in [4]. The objective is to specify a process, P, within a network as a tuple, (ϕ, A, C, ψ). Formally an assumption-commitment formula has the following form:

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

where P denotes a process and A, ϕ, ψ, C denote predicates. An A-C formalism given by Misra and Chandy requires that A and C are predicates whose values do not depend on any program variables. In general assumption and commitment reflect the communication interface between parallel components and do not refer to program variables of a process. Pre and post conditions do refer to these variables and facilitate reasoning about sequential composition and iteration. In this formalism, parallel composition rules can be formulated entirely

in terms of those predicates without any need for additional details about the implementation of the processes concerned.

Informally an Assumption-Commitment 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*

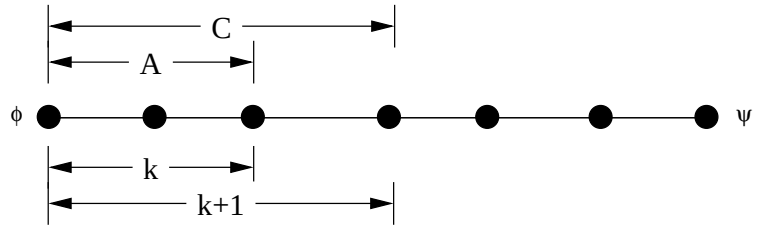

Fig. 1. *Assumption-Commitment*

An equivalent definition using induction (*Fig.* 1) follows:

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

denotes that,

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 (A expresses at any moment during an ongoing or terminated computation the properties that are assumed for its previous inputs), 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. The Assumption-Commitment paradigm provides an abstraction from the actual implementation of the environment, since it does not refer to any known processes in the environment. Assumption and Commitment are required to hold for both terminated and nonterminated computation.

4 Formalising Assumption-Commitment in ITL

4.1 Overview of Interval Temporal Logic

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.

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

Fig. 2. Syntax of ITL

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 $\Box$ extending the conventional logic to temporal reasoning. Expressions and Formulae are evaluated relative to the beginning of an interval.

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

- $\imath a : f$: 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 finite number of intervals such that for each of them f holds.

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

4.2 Application to Webservices

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 Webservices. For our purpose, we need to use a variant of the formalism proposed by Misra and Chandy 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*

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
$\textcircled{w} f$	$\widehat{=} \neg \bigcirc \neg f$	weak next
$\textcircled{i} f$	$\widehat{=} f$; *true*	some initial subinterval
$\boxed{i} f$	$\widehat{=} \neg(\textcircled{i} \neg f)$	all initial subintervals
$\textcircled{a} f$	$\widehat{=}$ *finite* ; f ; *true*	some subinterval
$\boxed{a} f$	$\widehat{=} \neg(\textcircled{a} \neg f)$	all subintervals

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 which services can be composed and once they are composed, it helps in validating the integrity of the composition at runtime.

The assertions required for formulating Assumption-Commitment 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.

Formally an assumption-commitment formula for a service S retains the following form:

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

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

A detailed treatment of Assumption-Commitment formalism in ITL follows in the next section.

4.3 An ITL formalization of Assumption-Commitment

A Service, S in ITL is expresses as a quadruple $(\omega, As, Co, \omega^{'})$ Validity of an A-C formula, analogous to that specified by Misra and Chandy,

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

and 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.

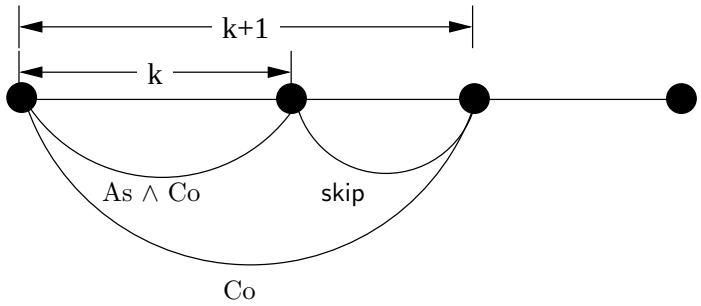

Fig. 3. *ITL representation of Assumption-Commitment*

Formally in ITL, the validity of the A-C representation above, has the following form :

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

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.* We define compositional rules using assumption - commitment for the 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. G represents the guard i.e the condition required to be true for the selection of a service from the network.

5.1 Sequential Composition

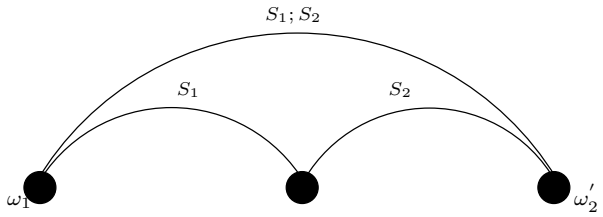

Fig. 4. *Sequential Composition*

We consider the sequential composition of two services. Services S_1 and S_2 can be specified in the assumption-commitment style of specification in ITL, as defined below,

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

In case of sequential composition, we require assumption and commitment to be of special types. Putting restrictions on the kind of assertions allowed for assumption and commitment, eliminates the overhead of requiring to prove the validity of additional proof obligations. We require that As and Co are respective fixpoints[1]of the ITL operators $\boxed{a}$ and *chopstar*. Assumption is a property of the environment, while commitment, a property of the system in response to the environment. For the commitment to hold over the interval defined by

[1] 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).

$S_1; S_2$, 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.

To illustrate the use of temporal assertions in sequential composition, we take the 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. Valid initial-final state predicates for the composition would therefore be:

`precondition:` $valid(ISBN) \wedge validCreditCard(cardNumber)$
`postcondition:` $cardBilled(cardNumber) \wedge bookShipped(shippingAddress)$

However, as we see from the above, these assertions do not make any statements about the credit card validity by the requestor throughout the composition and the assurance that the card will not be billed till the transaction is complete by the service provider. Assumption- commitment 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 requestor'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.

`assumption:` $\boxed{a}\, validCreditCard(cardNumber)$
`commitment:` $(\neg cardBilled(cardNumber))^*$

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 synchronously via message passing. Asynchronous message passing requires the addition of buffers to the synchronous picture, where buffers are modelled as services communicating synchronously with other services and services communicate with each other only via buffers. The specification, proof obligations and the compositional rule for services executing in parallel are as defined below:

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

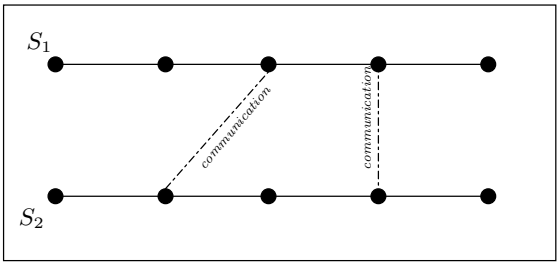

Fig. 5. *Parallel Composition*

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 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.

5.3 Iteration

Iteration, defines a service, executing repeatedly for a finite number of steps. The composition is analogous to a service executing "n" times in a sequential composition with itself. The specification, proof obligations and the compositional rule are defined below.

$$\frac{\begin{array}{lll}(As, Co) : \{\omega\}S\{\omega^{'}\} & \ldots(1)\\ As \equiv \boxed{a}\, As & \ldots(2)\\ Co \equiv Co^{*} & \ldots(3)\end{array}}{(As, Co) : \{\omega\}S^{*}\{\omega^{'}\} \ldots(4)}$$

5.4 Repeat-Until

This case is similar to Iteration, where the number of iterations depends on the truth value of the guard, G. The specification, proof obligations and the compositional rule are defined below.

$$\frac{\begin{array}{ll}(As, Co) : \{\omega\}S\{\omega^{'}\} & \ldots(1)\\ As \equiv \boxed{a}\, As & \ldots(2)\\ Co \equiv Co^{*} & \ldots(3)\\ \mathit{fin}\ \neg G \supset \mathit{fin}\ \omega^{'} & \ldots(4)\end{array}}{(As, Co) : \{\omega\}\mathit{Repeat}\ S\ \mathit{until}\ G\{\omega^{'}\} \ldots(5)}$$

5.5 if-then-else

We treat if-then-else as a sequential composition of three services where the second service in the sequence is a service, selected on the basis of a condition. The specification, proof obligations and the compositional rule are defined below.

$$
\begin{array}{ll}
(As, Co) : \{\omega_1\} S_1 \{\omega_1^{'}\} & \ldots (1) \\
(As, Co) : \{\omega_2\} S_2 \{\omega_2^{'}\} & \ldots (2) \\
(As, Co) : \{\omega_3\} S_3 \{\omega_3^{'}\} & \ldots (3) \\
(As, Co) : \{\omega_4\} S_4 \{\omega_4^{'}\} & \ldots (4) \\
As \quad \equiv \quad \boxed{a}\, As & \ldots (5) \\
Co \quad \equiv \quad Co^* & \ldots (6) \\
(\omega_1^{'} \wedge G \supset \omega_2) \vee (\omega_1^{'} \wedge \neg G \supset \omega_3) & \ldots (7) \\
(\omega_2^{'} \vee \omega_3^{'} \supset \omega_4) & \ldots (8) \\
\hline
(As, Co) : \{\omega_1\} S_1 \,;\, ((S_2 \wedge G) \vee (S_3 \wedge \neg G)) \,;\, S_4 \{\omega_4^{'}\} & \ldots (9)
\end{array}
$$

5.6 Choice

Let S represent the set of services then,

$S_1, S_2, \ldots, S_n \in S$

Let C represent a subset of S with cardinality m.

choice(C)$\stackrel{def}{=} C \subseteq S : |C| = m$

This is a non-deterministic choice of m processes from n processes

choice(C)$\stackrel{def}{=} C \subseteq S : |C| \geq m$

This is a non-deterministic choice of at least m processes from n processes

choice(C)$\stackrel{def}{=} C \subseteq S : |C| \leq m$

This is a non-deterministic choice of at most m processes from n processes

- sequence
  $\overset{i=m}{\underset{i=1}{;}} S_i, where\ S_i \in C$, the processes are composed in sequence as defined in section 5.1
- parallel
  $\overset{i=m}{\underset{i=1}{\|}} S_i, where\ S_i \in C$, the processes are composed in parallel as defined in section 5.2

6 Discussion

6.1 A Compositional Approach to Verification of Webservices

Traditional methods of verifying a composed system are non-compositional 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 apriori 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 Webservices 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.

6.2 Runtime Verification using AnaTempura

AnaTempura, 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.

7 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. [13] 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 [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 [8] 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 webservices. We believe that both specification and verification should be highly compositional allowing modular validation and verification to be performed. 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 plan to extend our work on verification of services and develop a practical approach using AnaTempura. Our next step would be to analysis realistic webservices using the theory proposed and provide an implementation framework for the same.

References

1. ITL 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. Jones 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. Willem-Paul de Roever et al. Concurrency Verification: Introduction to Compositional and Noncompositional Methods. Cambridge University Press, Cambridge, England, 2001.
7. Edsger W. Dijkstra. Solution of a problem in concurrent programming control. Number 8(9). CACM, 1965.
8. Intalio, Sun Microsystems, BEA Systems, SAP. *Web Service Choreography Interface (WSCI) 1.0 Specification*, 2002.
9. J.Misra and K.M.Chandy. Proofs of networks of processes. volume 7(7):417-426. IEEE Transactions on software Engineering, 1981.
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. Srini Narayan and Sheila A.Mcllraith. *Simulation, Verification and Automated Composition of Web Services*. Hawaii, USA, 2002. Eleventh International World Wide Web Conference.
14. OWL-S Coalition. OWL-S 1.0 (Beta) Draft Release. , 2003. *http://www.daml.org/services/owl-s/1.0/* .
15. 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*.
16. Mithun Sheshagiri, Marie desJardins, and Tim Finin. A Planner for Composing Services Described in DAML-S. In *Proceedings of the AAMAS Workshop on Web Services and Agent-based Engineering*, 2003.
17. T. Berners-Lee, J.Hendler and Ora Lassila. *The Semantic Web*. 2000.
18. Dan Wu, Bijan Parsia, Evren Sirin, James Hendler, and Dana Nau. Automating DAML-S Services Composition Using SHOP2. In *Proceedings of the 2nd International Semantic Web Conference*, 2003.