Chapter 5

TEMPORAL REASONING OF REACTIVE WEB SERVICES

Monika Solanki, Antonio Cau and Hussein Zedan.
Software Technology Research Laboratory, De Montfort University, Leicester, LE4 0GL, UK– monika@dmu.ac.uk, acau@dmu.ac.uk, zedan@dmu.ac.uk

1. WEB SERVICES AS REACTIVE SYSTEMS

Computing systems can be conceptually partitioned into two primitive categories: **Transformational** and **Reactive**. Transformational systems, as shown in Figure 5-1 are generally modelled by abstracting away the computations and specifying the system as an input-output function. The non-termination of a transformational system is usually considered a failure. Compilers, assemblers and routines in a library of mathematical functions are examples of transformational systems. The objective of Reactive systems[1] (D. Harel and A. Pnueli 1985) on the other hand is not necessarily terminating after producing some result, but maintaining an ongoing interaction with their environment and responding with appropriate actions to the external stimuli. When designing, describing and reasoning (Kim Sunesen 1998) about reactive systems, the focus is not just on what is computed but equally on how and when it is computed, in terms of interaction capabilities over time. Conventional examples of reactive systems include flight control systems, nuclear reactors, web applications, electronic games and touch screens. Reactive systems as illustrated in Figure 5-2 cannot be specified by a relation between initial and final states.

[1] The term was coined by Harel and Pnueli (D. Harel and A. Pnueli 1985). A brief but useful discussion can be found in (Harel and M. Politi 1998).

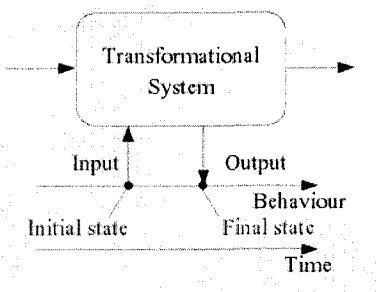

Figure 5-1 A simple transformational system

Although traditionally, Web services have been thought of as being information intensive, transformational programs, most useful Web services are in fact reactive systems. Examples include, web services deployed and composed as e-commerce applications, where an order once placed, can be cancelled, changed or put on hold because of unexpected conditions, anytime before its fulfillment. In certain cases a refund may also be requested later, if the service/product does not meet its specifications. In corporate e-business, it may not be a simple database query that generates a document, but an entire business process involving multiple partners. The final generation of the document may span several days. Web services deployed on wireless devices may take more than expected time to provide the requested service due to poor connection facilities.

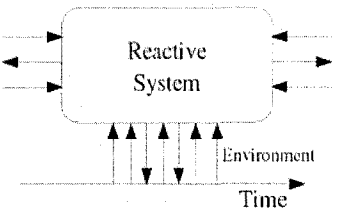

*Figure 5-2.*A Reactive system

Consider a typical example of a flight reservation service. The service provides results for a flight search and reserves tickets for the selected flight, thus changing the status of a seat from unbooked to booked i.e. transforming information by execution of a database query. However, the final selection of flight by a travel agent can span over an unlimited period of time, going

through several rounds of selection. A typical interaction is shown in Figure 5-3. The service may also exert control over the environment by terminating the user session after pre-specified time limits of inactive sessions. In case of flight search the database server itself is reactive as it allows the environment i.e. service requesters to ask queries. Further, once a flight has been booked, the agent also has the option of cancelling the booking within a stipulated time period.

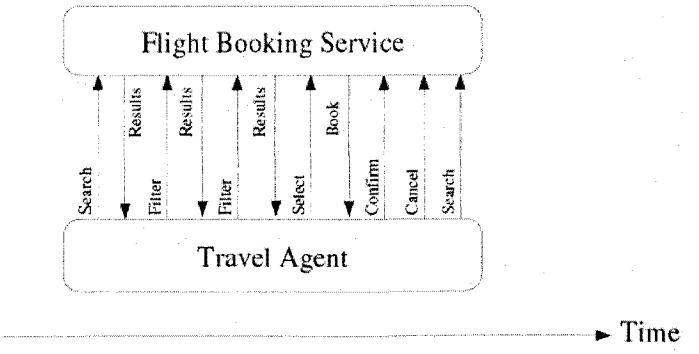

*Figure 5-3.*A Typical Flight Reservation Scenario

Further, service composition represent long running interactions between service requesters and providers that extend beyond single step execution of services. In order to correctly specify their behaviour, properties of services need to be expressed in a form that enables reasoning about their behaviour during such extended execution. Current XML-based and ontological specification standards for the description of service behaviour, do not have the capability to specify compositional properties. Languages like WSDL (Roberto Chinnic et al. 2005) and WSBPEL (Tony Andrews et al. 2003) provide an operational approach to service specification. They do not have the provision for specifying the conditions that restrict the execution of services to a limited set of valid behaviours. In other frameworks like OWL-S (The OWL-S Coalition 2004) and WSMO, specification of pre/post-conditions and effects contribute to some extent towards their behavioural specification. However they are limited to static behaviour descriptions in the sense that they are predicates required to hold only at the initial and final states.

The need for more expressive service specification also becomes evident, while reasoning about the composition of services and validation of the composition at runtime. Model checking (E.M. Clarke et al 1999) and theorem

proving are commonly used techniques for formal verification. In the context of analysing services and their composition at runtime, these techniques are not feasible due to the possible exponential growth in the number of reachable global states. In contrast to formal verification, practical validation techniques provide a mechanism to verify only properties which are of interest to the service requester or provider. Our notion of validation is different from the classical technique of "testing", generally associated with it. We believe, validation is a process of checking for inconsistent, redundant, incomplete or incorrect properties for a service. Properties are checked not for all possible behaviours (Shikun Zhou 2003) as in verification, but for a particular trace or execution of a service. As shown in our earlier work on service composition (M. Solanki et al, 2004), the objective of runtime validation is not to prove individual service implementation correct. It is to ensure that no undesirable behaviour emerges, when the service is composed with other services.

In this chapter, we propose a methodology to **compositionally** augment the semantic description of a reactive service, with **temporal** properties that provide the required support for reasoning about "ongoing" behaviour. The properties are specified in **Interval Temporal Logic** (ITL) (B. Moszkowski, 1986, 1994, 1996), our underlying formalism for reasoning about service behaviour over periods of time. These properties are specified only over observable behaviour, and do not depend on any additional knowledge about the underlying execution mechanism of the services. We present "TeSCO-S", a framework for enriching Web service interface specifications, described as OWL (Mike Dean and Guus Schreiber 2004) ontologies with temporal assertions. TeSCO-S provides an OWL ontology for specifying properties in ITL, a pre-processor, "OntoITL" for transforming ontology instances into ITL formulae and an interpreter, "AnaTempura" that executes and validates temporal properties in "Tempura", an executable subset of ITL.

2. A MOTIVATING EXAMPLE: AN ONLINE BOOKSTORE

An Online Bookstore as shown in Figure 5-5 is a sequential composition of four services: Book search, Book buy, Payment validation and Book delivery. Each of these services is a reactive service, as they continuously interact with the customer as illustrated in Figure 5-4. The e-Bookshop requires the customer to be registered with the service, in order to search or buy a book. The customer sends the ISBN number of the book to the Book search service, which returns a message with the search results. The

customer can continue searching for more books, always supplying the ISBN number or proceed to buy the book. The Book buying service, takes as input the list of books selected by the customer, the delivery address and the credit card details. The Card details and address are passed to the Payment validation service. If the card is validated, then depending on the amount paid and mode of delivery selected (standard or express), the book is arranged to be delivered to the customer. We informally define properties of the composition, some of which we formalise in the subsequent sections. We perceive Web services as black boxes and hence the properties strictly characterise the observable behaviour of services in the composition.

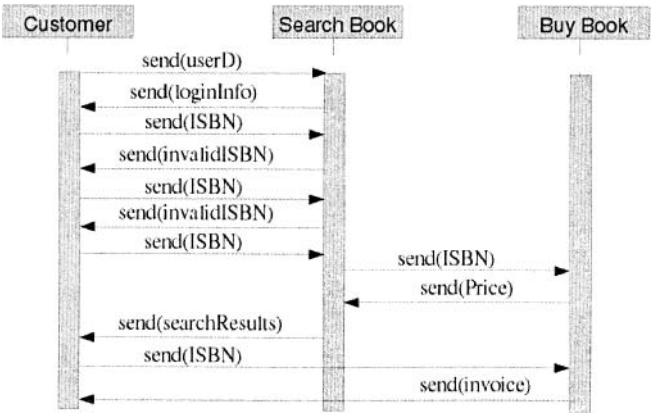

*Figure 5-4.*Interactions in an Online Bookstore

- At all times during the execution of the composed service, the customer is required to be a registered member of the e-Bookshop. This is a useful property to validate, when an inactive customer session is activated after a considerable period of time. Most services store customer registration details as session data, which is reset after a predefined period of inactivity.
- Once a customer starts searching for a book, the price of the book has to be constant till the search is over or if the customer buys the book, the price has to be constant till the book has been delivered to the customer.
- During the search, at any time if the customer sends an ISBN number, he gets back the search results, for the same ISBN number.

- Once a book or a list of books have been selected and ordered, the parameters of the book (title, language etc) should not change, till the book has been delivered to the customer.

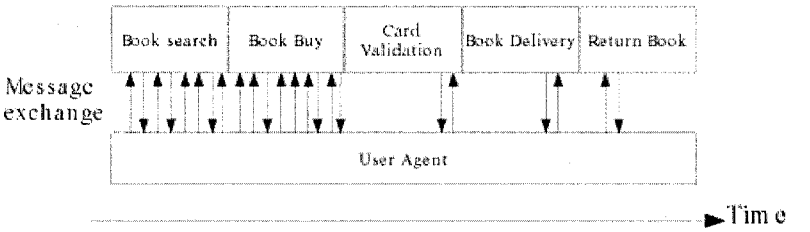

Figure 5-5. A Typical Book Buying Scenario

- In order to buy a book, the customer needs to have a valid credit card.
- Once the credit card has been validated, the e-Bookshop makes a commitment to deliver the book as per the delivery terms and conditions agreed with the customer.

We use the Online Bookstore as a running example throughout the chapter to explain various concepts

3. INTERVAL TEMPORAL LOGIC

ITL is an important class of temporal logic which was initially devised by Ben Moszkowski in the 1980's in order to model digital circuits (B Moszkowski, 1983). Later it was designed particularly as a formalism for the specification and design of software systems (B Moszkowski, 1995, 1994, 1996). ITL is an extension of classical first order logic especially designed for representing time dependent behaviour. It has proved to be an efficient formalism for specifying and reasoning about concurrently executing, real time critical systems.

3.1 Model

ITL is a linear-time temporal logic with a discrete model of time for both finite and infinite intervals. The model of behaviour used in ITL is quite natural. The idea is to describe the system of interest by taking a number of

"snapshots" at various points in time t_i, for $i \leq n$ and linking these snapshots together $(t_0 \ldots t_n)$. This link is the key notion in ITL and is called an "interval". Snap- shots define various relevant "states" for modelling the system and an interval is considered as an (in)finite, nonempty sequence of states $\sigma_0\sigma_1\cdots$

$$\sigma : \sigma_0\sigma_1\sigma_2\cdots$$

Each state represents a mapping from the set of variables *Var* and their values *Val.*

$$State: Var \rightarrow Val$$

The length $|\sigma|$ of a finite interval σ is equal to the number of states in the interval minus one. An empty interval has exactly one state and its length is equal to 0. The notation $\sigma_{i:j}$ denotes the subinterval of length *j-i* with states $\sigma_i, \sigma_{i+1}, \cdots, \sigma_j$

3.2 Syntax

The syntax of ITL is defined in Figure 5-6, where μ is an integer value, a is a static variable (does not 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
$e ::=$	$\mu \mid a \mid A \mid g(exp_1, \ldots, exp_n)$
	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 5-6.*Syntax of ITL

1. **Operators**:

ITL contains conventional propositional connectives such as $\wedge, \neg$ and first order ones such as $\forall, \exists$ and =. Extending the logic to temporal reasoning are operators like "; (chop)", "* (*chopstar*)" and "*skip*". Additional temporal operators defined in ITL include ○ (next) and □ (always).

2. **Expressions**:

Expressions are built inductively from variables, constants and functions as follows:

- Constant: μ

  A constant is denoted by a function without parameter. These are fixed

values
Examples: true, false, 2, 3, 5, [2, 3, 4, 5].

- Variables: A, B, C, . . . , a, b, c
  The value of a state variable can change within the interval, while the value of a static variable remains constant throughout the reference interval. Conventionally capital letters denote state variables, while small letters denote static variables. The letter v is used as a meta-variable in definitions to range over all variables.
- Function: $g(\exp_1, \cdots, \exp_n)$, where $n \geq 0$
  The function symbols include arithmetic operators such as +,-, *mod* and * (multiplication). Constants such as 0 and 1 are treated as zero place functions.
  Examples: *A + B, a-b, A + a, v mod C*
- $ia : f$: An expression of the form $ia : f$ is called a *temporal expression.* It returns a value a for which the formula f holds in the reference interval. If there is no such an a then $ia : f$ takes an arbitrary value from a's range.

Some examples of syntactically legal expressions are given below:

- $I + (\circ J) + 2$
  This expression adds the value of I in the current state, the value of J in the next state and the constant "2".
- I+ (○J) -○○(I)
  This expression adds the value of I in the current state to the value of J in the next state and subtracts the value of I in the next to next state from the result.

3. **Formulae**:

Formulae are built inductively from predicates and logical connectives as follows:

- Atomic formulae are constructed using relation symbols such as = and $\leq$.
  Examples: $e_0 \leq e_1$
- Logical connectives: $\neg f, f_1 \wedge f_2$ where f, f_1, f_2 are formulae.
- Universal Quantifier: $\forall v.f$
- Temporal Operators: skip, ";", "(chop) and "*" (*chopstar*) Examples:
  $$f_1; f_2, f*$$
  Some examples of syntactically legal formulae are given below:

- (J=2)○(K=4)
  This formula states that the value of *J* is "2" in the current state and the value of *K* is "4" in the next state.
- ○(□[I=2] ∧ ○□[J=2])
  The formula states that from the next state, the value of *I* would always be equal to "2" and the value of *J* in the next to next state will be equal to "2".

Many more examples can be found in (B. Moszkowski 1986).

3.3 Informal Semantics

Expressions and Formulae in ITL are evaluated relative to the beginning of an interval. Formulae with no temporal operators are called "state" formulae. With respect to an interval, a state formula is required to hold only at the initial state of that interval.

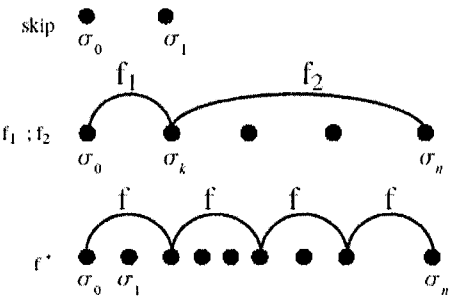

Figure 5-7. Pictorial illustration of ITL Semantics

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

- skip: unit interval (length 1).
  The formula skip has no operands and is true on an interval iff the interval has length 1 (i.e. exactly two states).

- $f_1; f_2$: A formula $f_1; f_2$ is true on an interval σ with states $\sigma_0 \cdots \sigma_{|\sigma|}$ iff the interval can be "chopped" into two sequential parts (i.e. a prefix and a suffix interval) sharing a single state σ_k for some $k \leq |\sigma|$ and in which the subformula f_1 is true on the left part $\sigma_0 \cdots \sigma_k$ and the subformula f_2 is true on the right part $\sigma_k \cdots \sigma_{|\sigma|}$.

- f^*: A formula f^* is true over an interval iff the interval can be chopped into zero or more sequential parts and the subformula f is true on each.

Figure 5-7 pictorially represents the semantics of *skip, chop and chopstar*. Some ITL formulae together with intervals which satisfy them are shown in Figure 5-8

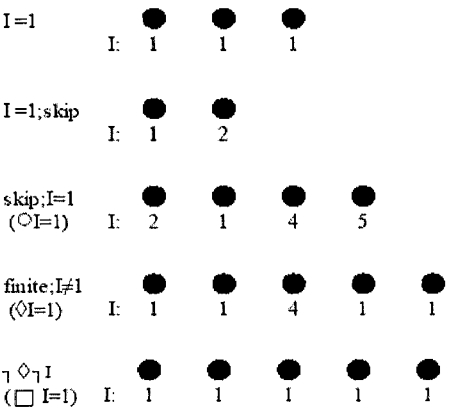

Figure 5-8. Some sample ITL formulae and satisfying intervals

true	$\widehat{=} 0 = 0$	true value
false	$\widehat{=} \neg true$	false value
$f_1 \vee f_2$	$\widehat{=} \neg(\neg f_1 \wedge \neg f_2)$	or
$f_1 \supset f_2$	$\widehat{=} \neg f_1 \vee f_2$	implies
$f_1 \equiv f_2$	$\widehat{=} (f_1 \supset f_2) \wedge (f_2 \supset f_1)$	equivalent
if g then f_1 else f_2	$\widehat{=} (g \wedge f_1) \vee (\neg g \wedge f_2)$	if-then-else
$\exists v \bullet f$	$\widehat{=} \neg\forall v \bullet \neg f$	exists

Figure 5-9. Non-temporal constructs

3.4 Derived Constructs

The following constructs can be derived from primitives of the logic. Non-temporal constructs are presented in Figure 5-9. Frequently used temporal modalities are represented in Figure 5-10. The formula "f" is used as a reference formula for defining the constructs.

$\bigcirc f$	$\widehat{=}$ skip ; f	next
more	$\widehat{=} \bigcirc true$	non-empty interval
empty	$\widehat{=} \neg more$	empty interval
inf	$\widehat{=} true\ ; false$	infinite interval
finite	$\widehat{=} \neg inf$	finite interval
$\Diamond f$	$\widehat{=} finite\ ; f$	sometimes
$\Box f$	$\widehat{=} \neg\Diamond\neg f$	always
$\Diamond\!\!\!\cdot f$	$\widehat{=} f\ ; true$	some initial subinterval
$\boxdot f$	$\widehat{=} \neg(\Diamond\!\!\!\cdot \neg f)$	all initial subintervals
$\Diamond\!\!\!\cdot f$	$\widehat{=} finite\ ; f\ ; true$	some subinterval
$\boxdot f$	$\widehat{=} \neg(\Diamond\!\!\!\cdot \neg f)$	all subintervals
halt f	$\widehat{=} \Box(\text{empty} \equiv f)$	terminate interval when
fin f	$\widehat{=} \Box(\text{empty} \supset f)$	final state
fin exp	$\widehat{=} \imath a:\ fin\ (exp = a)$	end value
keep f	$\widehat{=} \boxdot(\text{skip} \supset f)$	all unit subintervals
$\bigcirc exp$	$\widehat{=} \imath a:\ \bigcirc(exp = a)$	next value
$exp_1 \leftarrow exp_2$	$\widehat{=} finite \wedge (fin\ exp_1) = exp_2$	temporal assignment
$exp_1\ gets\ exp_2$	$\widehat{=} keep\ (exp_1 \leftarrow exp_3)$	gets
stable *exp*	$\widehat{=} exp\ gets\ exp$	stability
len(exp)	$\widehat{=} \exists I \bullet (I = 0) \wedge (I\ gets\ I + 1) \wedge I \leftarrow exp)$	interval length

Figure 5-10. Frequently used temporal abbreviations

3.5 Types in ITL

There are two basic inbuilt types in ITL. These are integers N and Boolean (true and false). In addition the executable subset of ITL (tempura) has basic types: integer, character, boolean, list and arrays. Further types can be built from these by means of X and the power set operator P (in a similar fashion as adopted in the specification language Z (M Imperato, 1991). For example the following introduces a variable x of type T.

$$(\exists x : T).f \stackrel{def}{=} \exists x.type(x,T) \wedge f$$

Here $type(x,T)$ denotes a formula that describes x to be of type T. Although this might seem to be a rather inexpressive type system, richer types can be added following that of (Spivey, 1996).

3.6 Formal Semantics

In this section we present the formal semantics of expressions (terms) and formulae in ITL. We define the data domain to be a set of integers denoted by Z. We assume "tt, ff" to represent the set of truth values. A state

(σ) is then a function mapping from variables Var to values in Z. We let Σ denote the set of all such functions,

$$\sigma_i \in \Sigma \hat{=} Var \to Z$$

Each n-ary function symbol g is associated with a total function

$$\hat{g} \in Z^n \to Z$$

Interpretations of n-ary relational symbols ($\hat{p}$) are similar but map to truth values.

$$\hat{p} \in Z^n \to tt, ff$$

Function symbols, e.g. + and −, and relation symbols, e.g. ≥ and =, are assumed to have their standard meanings. We define Σ^+ and Σ^ω to denote sets of finite and infinite intervals respectively. The relation

$$\sigma \sim_V \sigma'$$

is defined to be true iff the interval σ and σ', ($\sigma, \sigma' \in \Sigma^+ \cup \Sigma^\omega$) have the same length and agree on the behaviour of all variables except possibly the variable V.

3.6.1 Semantics of Expressions

The construct $\mathcal{E}_\sigma[\![exp]\!]$ denotes the function that defines the value in $\mathbb{Z}$ of the expression exp on the interval σ.

$$\mathcal{E}_\sigma[\![exp]\!] \in (\Sigma^+ \cup \Sigma^\omega) \to \mathbb{Z},$$

- $\mathcal{E}_\sigma[\![\mu]\!] = \sigma_0(\mu) = \mu.$
- $\mathcal{E}_\sigma[\![a]\!] = \sigma_0(a)$ and
  for all i s.t. $0 \le i \le |\sigma|, \sigma_i(a) = \sigma_0(a).$
- $\mathcal{E}_\sigma[\![A]\!] = \sigma_0(A).$
- $\mathcal{E}_\sigma[\![g(exp_1, \ldots, exp_n)]\!] = \hat{g}(\mathcal{E}_\sigma[\![exp_1]\!], \ldots, \mathcal{E}_\sigma[\![exp_n]\!]).$
- $\mathcal{E}_\sigma[\![\imath a\colon f]\!] = \begin{cases} \chi(u) & \text{if } u \neq \emptyset \\ \chi(Val_a) & \text{otherwise} \end{cases}$
  where $u = \{\sigma'(a) \mid \sigma \sim_a \sigma' \wedge \mathcal{E}_{\sigma'}[\![f]\!] = \text{tt}\}$

3.6.2. Semantics of Formulae

The construct $\mathcal{E}_\sigma[\![f]\!]$ denotes the function that defines the value in (tt, ff) of the formula f on the interval σ.

$$\mathcal{E}_\sigma[\![f]\!] \in (\Sigma^+ \cup \Sigma^\omega) \rightarrow \{tt, ff\},$$

- $\mathcal{E}_\sigma[\![p(exp_1, \ldots, exp_n)]\!] = \text{tt}$ iff $\hat{p}(\mathcal{E}_\sigma[\![exp_1]\!], \ldots, \mathcal{E}_\sigma[\![exp_n]\!])$.
- $\mathcal{E}_\sigma[\![\neg f]\!] = \text{tt}$ iff $\mathcal{E}_\sigma[\![f]\!] = \text{ff}$.
- $\mathcal{E}_\sigma[\![f_1 \wedge f_2]\!] = \text{tt}$ iff $\mathcal{E}_\sigma[\![f_1]\!] = \text{tt}$ and $\mathcal{E}_\sigma[\![f_2]\!] = \text{tt}$.
- $\mathcal{E}_\sigma[\![\forall v \cdot f]\!] = \text{tt}$ iff for all σ' s.t. $\sigma \sim_v \sigma'$, $\mathcal{E}_{\sigma'}[\![f]\!] = \text{tt}$.
- $\mathcal{E}_\sigma[\![\mathsf{skip}]\!] = \text{tt}$ iff $|\sigma| = 1$.
- $\mathcal{E}_\sigma[\![f_1 ; f_2]\!] = \text{tt}$ iff
  (exists a k, s.t. $\mathcal{E}_{\sigma_0 \ldots \sigma_k}[\![f_1]\!] = \text{tt}$ and
  ((σ is infinite and $\mathcal{E}_{\sigma_k \ldots}[\![f_2]\!] = \text{tt}$) or
  (σ is finite and $k \leq |\sigma|$ and $\mathcal{E}_{\sigma_k \ldots \sigma_{|\sigma|}}[\![f_2]\!] = \text{tt}$)))
  or (σ is infinite and $\mathcal{E}_\sigma[\![f_1]\!]$).

- $\mathcal{E}_\sigma[\![f^*]\!] = \text{tt}$ iff
  if σ is infinite then
  (exist $l_0, \ldots, l_n$ s.t. $l_0 = 0$ and $\mathcal{E}_{\sigma_{l_n} \ldots}[\![f]\!] = \text{tt}$ and
  for all $0 \leq i < n$, $l_i \leq l_{i+1}$ and $\mathcal{E}_{\sigma_{l_i} \ldots \sigma_{l_{i+1}}}[\![f]\!] = \text{tt}$.)
  or
  (exist an infinite number of l_i s.t. $l_0 = 0$ and
  for all $0 \leq i$, $l_i \leq l_{i+1}$ and $\mathcal{E}_{\sigma_{l_i} \ldots \sigma_{l_{i+1}}}[\![f]\!] = \text{tt}$.)
  else
  (exist $l_0, \ldots, l_n$ s.t. $l_0 = 0$ and $l_n = |\sigma|$ and

4. Compositional Reasoning for Web Services

Web services cannot exist in isolation. Most Web services interact with other services, users, devices or sensors to achieve a goal. The fundamental problem of composing specification of services, is to prove that a composite service satisfies its specification if all of the component services satisfy their specifications. For a compositional and modular specification of services, the description of interfaces between services and their environment is of utmost importance. The *interface* of a service provides the static/dynamic (logical) connection between the service and its environment. An interface description is a specification of those properties of a service that influences the overall behaviour of the composed system as well as those of the

individual services. Interface specification of reactive services cannot simply be described in terms of functions or relation on states, a more expressive representation format is needed.

4.1 Compositionality

Compositionality 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 (W.P. de Roever, 1985, 2001, J. Zwiers, 1989) is very important in computer science as it facilitates modular design and maintenance of complex systems following the *verify-while-develop* paradigm. Compositional proof techniques have the advantage that they allow the systematic top-down development of systems from their specifications. Compositionality is also a desired criterion for verification methodologies particularly for the development and analysis of large scale systems. The idea was first suggested by E. W. Dijkstra (E. W. Dijkstra 1965) in where he discusses hierarchical decomposition and verification of a given program on the basis of its subprograms, and formalised by (Floyd, 1967) where properties of a sequential program are derived from the properties of its atomic actions. For reasoning satisfactorily about composed system, systems and their components are specified using **assertional** specifications i.e. *state predicates*, only over their observable behaviour.

4.2 Applying the Assumption-Commitment Paradigm to Web Services

For the development of a compositional framework that allows the specification and validation of services and their composition, we choose the Assumption-Commitment paradigm. The objective of an **Assumption-Commitment** style of specification is to specify a process within a network. In its most general form Assumption-Commitment (P. K. Pandya 1990, Qiwen Xu and Mohalik Swarup, 1998) reasoning, allows the verification of a service under the assumption that the environment behaves in a certain way. The Assumption-Commitment style of specification has been applied extensively as a proof technique to networks of processes executing concurrently via synchronous message passing in a seminal work by (J. Misra and K.M. Chandy 1981).

In our earlier work on service composition, we have shown the power of assumption-commitment style of specification for compositional reasoning of ongoing service behaviour. We have proposed a methodology (Solanki et

al. 2004) to augment the specification of a service, with properties that are temporal and compositional, called *assumption* and *commitment.* Assumption-Commitment properties are specified only over observable behaviour, and do not depend on any additional knowledge about the underlying execution mechanism of the services. Interestingly, Interval Temporal Logic, our underlying formal framework can be used both for establishing the validity of the behaviour of a service and for proving the soundness of the compositional rules.

The assumption-commitment specification can be thought of as a pair of predicates (As, Co) where the assumption As specifies the environment in which the specified service is supposed to run, and the commitment Co states the requirement which any correct implementation of the service must fulfill whenever it is executed in an environment that satisfies the assumption. Since we are interested in the observable, ongoing behaviour of services, we model assumption-commitment as temporal properties defined over their interface specification.

4.3 An ITL Formalisation of Assumption-Commitment

A service, S, in ITL is expressed as a quadruple

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

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
ω' : state formula about final state

Figure 5-10. Frequently used temporal abbreviations

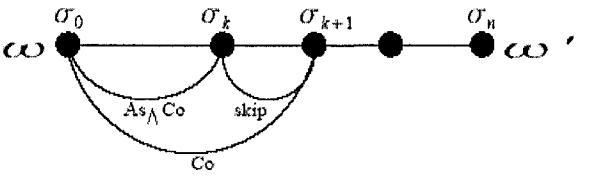

Figure 5-11. ITL representation of Assumption-Commitment

Formally in ITL, the validity of the Assumption-Commitment representation as illustrated in Figure 5-11 has the following form:

$$(As, Co) : \{\omega\} S \{\omega'\} \stackrel{def}{=} \omega \wedge S \supset (\boxdot(\mathsf{empty} \vee ((As \wedge Co); \mathsf{skip}) \supset Co \wedge_{fin} \omega'))$$

We have also proposed compositional proof rules based on assumption-commitment properties that allow validation of ongoing behaviour of services. Keeping in perspective the e-Bookshop service which is sequentially composed, we present the rules here for sequential composition.

We consider the sequential composition (ref. Figure 5-12) of two services, S_1 and S_2. For a detailed explanation of the rules and its proof obligations, the interested reader is referred to (Solanki et al. 2004).

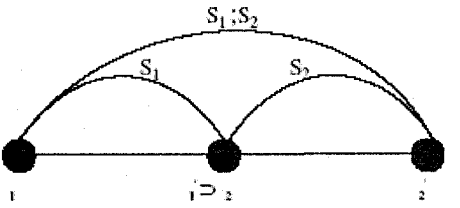

Figure 5-12 Sequential Composition

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

5. Formalisation of the Online Bookstore

We now formalise some of the interesting properties of the e-Bookshop service from section 2.

- At all states ($\sigma_0 \ldots \sigma_l$) during the execution of the composed service, the customer is required to be a registered member of the e-Bookshop.

$$\Box(isRegistered(userID))$$

- Once a customer starts searching for a book, the price of any book returned as a result has to be constant till the search is over or if the customer buys the book, the price has to be constant till the book has been delivered to the customer i.e. the price of the book has to be constant at all states ($\sigma_0 \ldots \sigma_l$).

$$\Box(isNotChanged(bookPrice))$$

- During the search ($\sigma_0 \ldots \sigma_m$), at any state if the customer sends an ISBN number, he gets back the search results, for the same ISBN number in the next state.

$$\Box((searchBook(ISBN)) \supset (searchResults(ISBN)))$$

- Once a book or a list of books have been selected and ordered, the parameters of the book (title, language etc) should not change, till the book has been delivered to the customer ($\sigma_m \ldots \sigma_l$).

$$\Box(isBook(selectedBook))$$

- In order to buy a book, the customer needs to have a valid credit card. that stays valid atleast till the book has been delivered to the customer ($\sigma_m \ldots \sigma_l$).

$$\Box(validCard(userID, cardNumber))$$

- Once the credit card has been validated, the e-Bookshop makes a commitment to deliver the book as per the delivery terms and conditions agreed with the customer ($\sigma_n \ldots \sigma_l$).

$$(finvalidCard(UserID, CardNumber))(DeliveryPeriod = CalculatedDays)$$

For sequential composition of services, the proof obligations require that we choose Assumption-Commitment properties of the form:

$$\vdash As \equiv \boxdot As$$
$$\vdash Co \equiv Co^*$$

We now define the assumption and commitment properties required to hold for the composition defined between states ($\sigma_0 \ldots \sigma_5$). Keeping in perspective the nature of properties, we informally define the assumption as,

At all states during the execution of the composed service, the customer is required to be a registered member of the e-Bookshop.

We define the corresponding commitment as

At all states during the execution, the e-Bookshop allows registered users to search and buy a book.

It is worth noting that these properties are specified as part of the behavourial specification of the e-Bookshop as well as the Customer. They are however required to be validated by the e-Bookshop. Formalising the above properties,

$$\boxdot(isRegistered(userID))$$
$$(validCustomer(userID))^*$$

For the composition between states σ_1 and σ_5, we define an additional commitment while keeping the assumption same,

Once a customer is returned the results of search, the price of book(s) selected should remain constant till the user finishes all transaction.

Formalising the above,

$$(unchangedPrice(userID, ISBN))^*$$

6. SEMANTIC ANNOTATION OF TEMPORAL SPECIFICATION: TESCO-S

Web services are discovered and composed based on the declarative specification of their interfaces as exposed by service providers in service registries or repositories. Temporal properties for services, need to be made a part of this declarative specification. In the context of temporal properties and Web services, the notion of "Temporal" can be interpreted in terms of the following two intuitive contexts:

- **Time-related properties of Web services**: expressing facts about dates (calendar) of events ("Order placed on 4th July"), duration of activities ("Shipping the product takes 24 hrs once an order is received") and absolute time i.e. clock ("Confirmation of a Shipped good will be sent out at 9.00 a.m. IST"). The vocabulary to describe these concepts include time as a first class citizen as part of their syntactic and semantic representation.
- **Behaviour-related properties of Web services**: expressing facts about ordering of services ("Check the credentials of the supplier, **before** placing an order "), constraints during service execution ("Do not modify a submitted order **while** the transaction is in progress", "**As long as** the supplier continues proves the authenticity of his goods, we shall continue to place orders with him.").

When describing temporal properties of services at a declarative level, we focus on the second notion i.e. reasoning about behaviour of services relative to time. The objective of declarative representation of temporal properties and constraints is to enable their automated reasoning and further their runtime validation for automated discovery, composition and execution of services. In the case of services that are semantically described, an important part of this effort is the development of representative ontologies of the most commonly used domains.

TeSCO-S (Temporal SemantiCs for OWL enabled Services) is a framework for semantically annotating and validating Web service specifications with temporal properties, defined using ITL and its executable subset "Tempura". The objective is:

- to provide an ontology for service providers to declaratively specify temporal properties in ITL.
- to provide a pre-processor for service requesters/composing middleware/software agents to process the declarative markup of properties and transform them into concrete ITL/Tempura formulae.
- to provide an execution engine for the generated tempura formulae, which can be used to validate properties about the service as well as perform runtime validation of assumption - commitment properties for service composition.

The semantics of the formulae and expressions modeled using TeSCO-S are the semantics as defined in ITL and implemented in its executable subset Tempura. TeSCO-S uses OWL as the ontology representation language. The choice of OWL as a representation format over XML is motivated by two objectives: (a) Our ultimate goal is to be able to automate reasoning about ITL formulae and expressions. (b) we want to be able to seamlessly use the ontology within standrads like OWL-S for services. Tools for reasoning about ITL-Tempura ontology, can be integrated with automated reasoning tools for services specified in OWL. For realising the objectives highlighted above, TeSCO-S includes the following components:

- An OWL ontology for first order formulae, expressions and temporal constructs as defined in ITL and Tempura.
- A pre-processor that transforms ontological representations of ITL and Tempura constructs defined in the ontology above to concrete formulae and expressions.
- An interpreter,"AnaTempura" that provides execution support for Tempura.

The following sections present a detailed discussion of each of these components.

6.1 The ITL-Tempura Ontology

The objective of the ITL-Tempura ontology is to express the syntactical framework of ITL and Tempura, as concepts and properties in OWL. ITL is very expressive and provides a number of primitive and derived constructs for the specification of a wide variety of temporal assertions. We have restricted the ontology to only a specific set, which we believe will be most useful and sufficient to express the kind of properties that most service providers would want to expose. On the other hand, the ontology itself is very modularly structured to enable future extensions. As discussed in

section (3), the syntax of ITL is defined primarily by Expressions and Formulae. Expressions can be of various types for e.g. static and state variables, functions, and constants. Similarly formulae can be subclassed as being atomic: e.g. " ",composite: e.g. " $f_1 f_2$ " and predicates: e.g. " *isRegistered*(*userID*) " amongst others. Expressions and Formulae in the ontology are built incrementally. The root class of all Formulae is "Formula", while that of Expressions is "Expression". Formula has several subclasses such as "Atomic", "Composite" and "Prefixed" amongst others. "TempuraFormula", defines formulae specified in Tempura and which can be executed by AnaTempura. "Operator" denotes the kind of operators that can be used with formulae and expressions. Classes have properties and restrictions associated that define the kind of parameters that are required to build the expression or formula. Properties provide the link between expressions/formulae and operators. We follow an incremental approach to building ontology instances using the ITL-Tempura ontology as shown in the e-Bookshop example presented in secton 6.5. The modular approach to building ITL and Tempura formulae allows reusability of formulae and expression instances between ontologies. We use the Protege OWL plugin for modelling the ontology.

Figure 5-13 shows how formulae and expressions are structured. A complete description of the ontology is beyond the scope of the paper. A graphical and hierarchical representation of the classes in the ontology can be found at (Solanki 2005). The complete ontology itself can be found at (Solanki 2005).

ITL-Tempura Ontology::=	Formula | Expressions | TempuraConstuct Connective | Operator | Quantifier
Formula::=	Atomic | TempuraAtomic | Equality | Composite | CompositeWithExpressions | Len | Negated | Prefixed | PrefixedWithExpressions Predicate | Quantified | Suffixed |
Expression::=	StateVariable | StaticVariable | Constant | Function | CompositeExpresions | MathFunc | NextExpression | PrefixExpression
Operator::=	EqualityOperator | TemporalOperator
TemporalOperator::=	InfixOpeartor | PrefixOperator | SuffixOperator

Figure 5-13 Primitives for the ITL-Tempura Ontology

6.2 OntoITL: A Pre-processor for Temporal Ontologies

So far, we have seen how ITL formulae and expressions can be modelled using the ITL-Tempura ontology. This enables service providers to specify temporal constraints as part of their service specification. In order to interpret this semantic markup of temporal properties, a utility is needed to generate concrete formulae and expressions from the OWL representation. The idea behind providing such a tool is to automate the process of generating, interpreting and analysing temporal properties of services. Service requestors and composers can use the tool to extract temporal properties that they would like to validate, while interacting with the service. At runtime, the properties are monitored against the behaviour of the interacting services.

OntoITL is a pre-processor that generates concrete ITL and executable Tempura formulae from instance ontologies built using the ITL-Tempura Ontology. The instances are defined using the core ontology as described in Section 6.1 or from ontologies that import these instances. It provides as output, complete information about instances of State and Static variables, Expressions, Formulae and Temporal Formulae modeled in the ontology. An output of the pre-processor for properties of the e-Bookshop, modeled using the ITL-Tempura Ontology and as explained in section 6.5 is shown in the Figure 5-16

OntoITL takes as input, the instance ontology in OWL for a formula or a set of formulae. It then generates ITL/Tempura formulae keeping the syntactical structure of the formula intact. OntoITL offers several options to store the generated ITL and Tempura formulae. It also provides the facility to directly pass the tempura formula to the AnaTempura interpreter, that executes the formulae and validates temporal properties. Alternatively, OntoITL stores the generated outputs in files that can be executed via the Tcl/Tk interface of AnaTempura as discussed in section 6.3.

6.3 AnaTempura: Validation of Tempura Specification

AnaTempura (available from (A. Cau, 2005)), which is built upon C-Tempura, is an integrated workbench for the runtime verification of systems using ITL and its executable subset Tempura. AnaTempura provides

- specification support
- verification and validation support in the form of simulation and runtime testing in conjunction with formal specification.

An overview of the run-time analysis process in AnaTempura is depicted in Figure 5-14.

Figure 5-14 The Analysis Process

There are two ways of validating properties via AnaTempura:

- Concrete Tempura formulae generated by the OntoITL pre-processor are directly passed to AnaTempura. The results of the validation and execution are returned to OntoITL for display.
- Concrete Tempura formulae generated by the OntoITL pre-processor are stored in files for validation at a later stage. The results of the validation and execution can be displayed via the Tcl/Tk interface of AnaTempura.

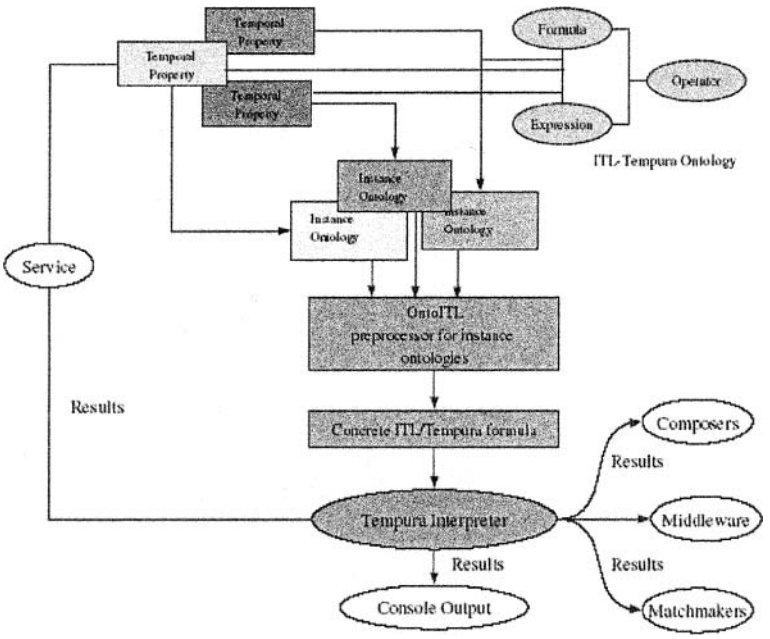

Figure 5-15 General Architecture for Web services

AnaTempura generates a state-by-state analysis of the system behaviour as the computation progresses. At various states of execution, values for variables of interest are passed from the system to AnaTempura. The Tempura properties are validated against the values received. If the

properties are not satisfied AnaTempura indicates the errors by displaying what is expected and what the current system actually provides. The approach goes beyond a "keep tracking" approach, i.e. giving the running results of certain properties of the system, by not only capturing the execution results but also comparing them with formal properties. The general architecture that employs AnaTempura for validation of service properties is shown in Figure 5-15.

The validation results of the instance-ontology-formulae, generated from the TeSCO-S framework, can be returned to the composing agents, the middleware or to the service requestor depending on the design of the service composition.

6.4 Validating the Customer: e-Bookshop Composition

We have validated the assumption-commitment properties of the e-Bookshop as formalised in section 5.1.

We adopt the second approach to validating properties as mentioned in section 6.3. The property is extracted as a tempura formula, from its ontological representation using the OntoITL pre-processor and stored in a file. At the initial state, the customer registers using his login details[2]. The login details are set for the customer session and passed to AnaTempura. As illustrated in the Figure 5-16 for each phase of the composition (search, buy etc.) and for every interaction between the e-Bookshop and the customer, at all states, the property is validated.

Tempura interpreter validates the property against the values set in the session for that state. We have developed a minimalistic GUI for dislaying the results of the property validation. The blue circle indicates that a property holds for that state, while a red circle indicates that a property has been violated. In the example shown, a "1" indicates the first service in the composition i.e. the "Book Search", while a "2" indicates the second service i.e. the "Book Buy". If the values in the session are found to be reset and do not match the ones passed to the interpreter in the initial state, a warning message is sent to the e-Bookshop as indicated by the red circle. It is worth noting that the interpreter only validates the properties of interest. It does not define the behaviour of the service in case the properties are not satisfied.

[2]For practical purposes, we do not model the registration process over an interval, although this may well be the case if the user enters incorrect login details, and takes several attempts to correct login.

This is a design decision that has to be taken before the composition is realised.

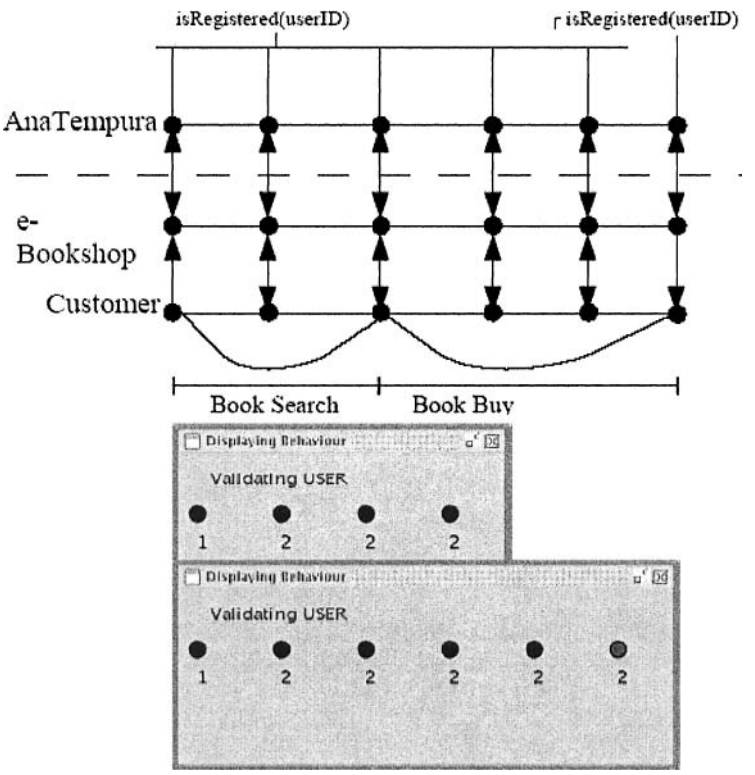

Figure 5-16 Validating the customer -e-bookshop composition

6.5 Specifying Properties in the ITL-Tempura Ontology

In this section, we model some interesting properties of the e-Bookshop service 5 using the ITL-Tempura ontology. For the sake of brevity in representation we model them as A-Box representations.

Recalling the definition of a composite formula,

Composite ô Formula ó (∀ hasPrefixedSubFormula.Formula) ó (∀ hasSuffixedSubFormula.Formula) ó (=1 hasInfixOperator.Operator) ó (=1 hasPrefixedSubFormula.Formula) ó (=1 hasSuffixedSubFormula.Formula)

We choose the following properties from the e-Bookshop example

Property (1): During the search, at any state if the user sends an ISBN

number, he gets back the search results, for the same ISBN number in the next state.

$((searchBook(ISBN)) \supset (searchResults(ISBN)))$

We define the properties as assertional axioms (ABox) in Description Logic. We build the formula incrementally as shown below:

ABox representation of Property (1):

ISBN:StateVariable, P1:Predicate, P2:Predicate
(P1, searchNook):hasName, (P1, ISBN):hasExpressionList
(P2, searchResults):hasName, (P2, ISBN):hasExpressionList
PR1:Prefixed, (PR1, Next):hasPrefixOperator, (PR2, P2):hasSubFormula
C1:Composite, (C1, Imp):hasInfixOperator
(C1,P1):hasPrefixedSubFormula, (C1, PR1):hasSuffixedSubFormula
PR2:Prefixed, (PR2, Always): hasPrefixOperator, (PR2, C1):hasSubFormula

Property (2): Once the credit card has been validated, the e-Bookshop makes a commitment to deliver the book as per the delivery terms and conditions agreed with the user.

$(finvalidCard(UserID, CardNumber))(DeliveryPeriod = CalculatedDays)$

ABox representation of Property (2):

UserID:StateVariable, CardNumber:StateVariable
DeliveryPeriod:StateVariable, CalculatedDays:StateVariable
P1:Predicate, (P1, validCard):hasName, (P1,
UserID,CardNumber)):hasExpressionList
PR1:Prefixed, (PR1, fin):hasPrefixOperator, (PR2, P1):hasSubFormula
EQ1:Equality, (EQ1, Equals):hasEqualityOperator, (EQ1,
DeliveryPeriod):hasPrefixExpression
(EQ1, CalculatedDays):hasSuffixExpression
C1:Composite, (C1, Chop):hasInfixOperator
(C1,P1):hasPrefixedSubFormula, (C1, EQ1):hasSuffixedSubFormula

7. CONCLUSIONS

From a historical perspective, research on Web services was initiated with a focus on automating business process composition within different enterprises. Such coordinations are long-lived processes and may last from a few minutes to a few months. An extensive review of state-of-the-art research in the domain of Web service composition reveals that current interface specification approaches do not provide capabilities to expose the

reactive aspect of Web service behaviour. Based on service interfaces definitions (Roberto Chinnic et al 2005) and message exchange protocols (Martin Gudgin et al.2003), standards have been proposed for specifying composite services, by defining declaratively, their data and control flows. BPEL4WS (Tony Andrews et al. 2003) provides distinct constructs for specifying abstract and executable processes. BPEL, however does not prevent complex computation from being included in an abstract process, thus revealing implementation details.

Within the context of semantic Web services frameworks like OWL-S and WSMO, specification of pre/post-conditions and effects contribute to some extent towards their behavioural description. However they are limited to describing transformational behaviour. There is no support available for describing and reasoning about changes over time. This is due to the lack of explicit modelling of "states" in these languages. Rule languages for the web include RuleML and within the context of semantic web, initiatives such as SWRL (Ian Horrocks et al. 2003) and DRS (Drew McDermott and Dejing Dou 2002). These approaches are limited to describing only certain kinds of properties. The expressivity of the languages is restricted to specifying static rules and constraints. There are no constructs available for specifying ongoing behavioural semantics or temporal properties of services. Other related work in this area is mostly concerned with representation of time as a first-class citizen, (Feng Pan and Jerry R. Hobbs 2004, F. Bry and S. Spranger 2003) i.e. reasoning about time points, complex time intervals, calendars and durations.

For dynamic composition of services, compositional properties need to be abstracted at a level where service requesters, providers, composing engines and matchmakers can discover these properties of services. Assumption-Commitment properties can be suitably specified in any service description language, rich enough to capture the underlying expressiveness of these properties. In this chapter, we provide a modular approach, TeSCO-S, to building and executing temporal properties of services, with interfaces described as OWL ontologies. TeSCO-S is based on Interval Temporal Logic (ITL) and Tempura, its executable subset. Our pre-processor "OntoITL" enables transformation of the bulky XML representation of temporal properties into concrete ITL and Tempura formulae, that can be handled readily by AnaTempura. The ontology within the TeSCO-S framework can be used by service providers to describe temporal capabilities of services. Service requestors and composing agents can use "OntoITL" and AnaTempura for on-the-fly transformation and validation of these temporal properties. The ontology provides constructs not only for specifying temporal expressions and formulae, but general first order predicates and formulae as well. It can therefore, also be used to specify pre-

conditions/post-conditions and effects in frameworks like OWL-S and WSMO. Ongoing work in TeSCO-S is providing reasoning support over temporal ontologies and tools for exploiting ITL formulae to build temporal ontologies. It is planned to have a protege plugin for defining temporal ontologies, that could be used along with the OWL-S editor for modelling OWL-S services.

8. QUESTIONS FOR DISCUSSIONS

Beginner:
1. What are the main categories under which computing systems can be partitioned?
2. What are the characteristics of reactive systems?
3. How does temporal logic help in formalising system behaviour?

Intermediate:
1. Discuss why Web services should be modelled as reactive systems.
2. What properties of a dynamically composed service can be formalised using temporal logic?
3. Discuss why the notion of Compositionality is important while defining composition of services.
4. Why should temporal properties of services be modelled as ontologies?
5. How does a service composition benefit from runtime validation of desired properties?

Advanced:
1. Discuss how properties of a holiday booking service can be formalised using Interval Temporal Logic.
2. How can properties of the holiday booking service be expressed using the ITL-Tempura ontology?
3. Identify assumption-commitment properties for the holiday booking services.

9. SUGGESTED ADDITIONAL READING

- Monika Solanki and Antonio Cau and Hussein Zedan. Introducing Compositionality in Web Service Descriptions. In Proceedings of the 10th International Workshop on Future Trends in Distributed Computing Systems - FTDCS 2004, Suzhou, China, May 26-28 2004. IEEE Computer Society Press.

- Antonio Cau. ITL and (Ana)Tempura Home page on the web. http://www.cse.dmu.ac.uk/~cau/itlhomepage/itlhomepage.html.
- Z. Manna and A. Pnueli.The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer-Verlag, New York, 1991.
- B. Moszkowski. Executing temporal Logic Programs. Cambridge University Press, Cambridge, England, 1986.

10. REFERENCES

F. Bry and S. Spranger (2003). Temporal constructs for a web language.

A. Cau, (2005). ITL and (Ana)Tempura Home page on the web. http://www.cse.dmu.ac.uk/~cau/itlhomepage/itlhomepage.html.

Antonio Cau and Hussein Zedan (1997). Refining interval temporal logic specifications. In ARTS, pages 79–94, 1997.

Roberto Chinnic, Hugo Haas, Amy Lewis, Jeans Jacque Moreau, David Orchard, and Sanjiva Weerawarana (2005). Web services description language (WSDL) version 2.0 part 1: Core language w3c working draft 3rd August, 2005. http://www.w3.org/TR/2005/WDwsdl20-20050803/.

W.P. de Roever (1985). The quest for compositionality a survey of assertion based proof systems for concurrent programs. In Neuhold EJ, editor, Proc of the IFIP conference: the role of abstract models in computer science,, Vienna. North Holland, Amsterdam.

W. P. de Roever et al (2001). Concurrency Verification: Introduction to Compositional and Noncompositional Methods. Cambridge University Press, Cambridge, England, 2001.

E. W. Dijkstra (1965). Solution of a problem in concurrent programming control. Commun. ACM, 8(9):569.

E. W. Dijkstra (1976). A Discipline of Programming. PrenticeHall.

Jurgen Dinge (2000). Systematic parallel programming. PhD thesis, Carnegie Mellon University.

Frank Leymann, IBM Software Group. Web Services Flow Language (WSFL) Version 1.0, 2001.

Drew McDermott and Dejing Dou (2002). Representing Disjunction and Quantifiers in RDF Embedding Logic in DAML/RDF. In ISWC2002. 1st International Semantic Web Conference, 2002.

E.M. Clarke and O. Grumberg, and D. A. Peled (1999). Model Checking. The MIT Press, Cambridge, Massachusetts.

R. W. Floyd. Assigning meaning to programs (1967). In Symposium in Applied Mathematics, volume 19, pages 19–31. American Mathematical Society, 1967.

Martin Gudgin, Marc Hadley, Noah Mendelsohn, JeanJacques Moreau, and Henrik Frystyk Nielsen (2003). SOAP Version 1.2 Part 1: Messaging Framework W3C Recommendation 24 June. http://www.w3.org/TR/soap12part1/.

D. Harel and A. Pnueli. (1985) On the development of reactive systems, pages 477–498. SpringerVerlag New York, Inc., New York, NY, USA .

D. Harel and M. Politi.(1998). Modeling Reactive Systems with Statecharts: The STATEMATE Approach. McGrawHill.

The Rule Markup Initiative. http://www.dfki.unikl.de/ruleml/.

C.A.R Hoare. An axiomatic basis for computer programming. Comm. ACM, 12 (1969) 576-580, 583, 1969.

Ian Horrocks, Peter F. PatelSchneider, Harold Boley, Said Tabet, Benjamin Grosof, Mike Dean (2003). SWRL: A Semantic Web Rule Language Combining OWL and RuleML . Technical report, University of Manchester, Version 0.5 of 19 November.

M Imperato (1991). An introduction to Z. ChartwellBratt, 1991.

Z. Manna and A. Pnueli (1991). The Temporal Logic of Reactive and Concurrent Systems: Specification. SpringerVerlag, New York.

Zohar Manna and Amir Pnueli .(1993) Models for reactivity. Acta Inf., 30(7):609–678.

Mike Dean and Guus Schreiber (eds.) 2004. OWL Web Ontology Language Reference, 10 February 2004. http://www.w3.org/TR/owlref/.

J. Misra and K.M. Chandy (1981). Proofs of networks of processes. In IEEE Transactions on Software Engineering, volume 7(7):417426.

Monika Solanki and Antonio Cau and Hussein Zedan (2003). Introducing compositionality in Webservice Descriptions. In Proceedings of the 3rd International Anwire Workshop on Adaptable Service Provision, Paris, France, 2003. SpringerVerlag.

Monika Solanki and Antonio Cau and Hussein Zedan (2004). Introducing Compositionality in Web Service Descriptions. In Proceedings of the 10th International Workshop on Future Trends in Distributed Computing Systems FTDCS 2004, Suzhou, China, May, 2004. IEEE Computer Society Press.

B Moszkowski (1983). Reasoning about Digital Circuits. PhD thesis, Department of Computer Science, Stanford University

B. Moszkowski (1986). Executing temporal Logic Programs. Cambridge University Press, Cambridge, England.

B. Moszkowski (1994). Programming Concepts, Methods and Calculi, IFIP Transactions, A-56., Some Very Compositional Temporal Properties, pages 307–326. Elsevier Science, B. V., NorthHolland, 1994.

B. Moszkowski (1995). Compositional reasoning about projected and infinite time. In Proceedings of the First IEEE Int'l Conf. on Engineering of Complex Computer Systems (ICECCS'95). In , pages 238245. IEEE Computer Society Press.

B. Moszkowski (1995). A temporal logic for multilevel reasoning about hardware. IEEE Computer, pages 10–19.

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

B. Moszkowski (1996). Using temporal fixpoints to compositionally reason about liveness. In He Jifeng, John Cooke, and Peter Wallis, editors, BCSFACS 7th Refinement Workshop, electronic Workshops in Computing. "SpringerVerlag and British Computer Society", London.

Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon (2004). Web Services Choreography Description Language Version 1.0: W3C Working Draft 17 December.

Feng Pan and Jerry R. Hobbs (2004). Time in OWLS. In Proceedings of AAAI Spring Symposium Series on Semantic Web Services, 2004.

P. K. Pandya (1990). Some comments on the assumptioncommitment framework for compositional verification of distributed programs. In REX workshop: Proceedings on Stepwise refinement of distributed systems: models, formalisms, correctness, pages 622–640, New York, NY, USA. SpringerVerlag New York, Inc.

Satish Thatte. XLANG: Web Services for Business Process Design, 2002.

Amazon Web Service. www.amazon.com.

Monika Solanki. (2005) A Graphical representation of Class Hierarchies in the ITLTempura Ontology. http://www.cse.dmu.ac.uk/~monika/TeSCOS/OntoITL.jpg.

Monika Solanki. (2005) An Ontology for ITL and Tempura. http://www.cse.dmu.ac.uk/~monika/TeSCOS/OntoITL.owl.

Monika Solanki, Antonio Cau, and Hussein Zedan (2004). Augmenting semantic web service descriptions with compositional specification. In Proceedings of the 13th international conference on World Wide Web, pages 544–552. ACM Press.

J. Michael Spivey (1996). Richer types for Z. Formal Asp. Comput., 8(5):565–584, 1996.

The protege ontology editor and knowledge acquisition system. http://protege.stanford.edu/index.html.

Ketil Stölen (1990). Development of parallel programs on shared datastructures. Technical report, Department of Computer Science, University of Manchester.

Kim Sunesen (1998). Reasoning about Reactive Systems. PhD thesis, BRICS, Department of Computer Science University of Aarhus.

The OWL-S Coalition, (2004). OWLS 1.1 Release. http://www.daml.org/services/owls/1.0/.

Tony Andrews et al. (2003) Business Process Execution Language for Web Services, Version 1.1, 2003. http://www106.ibm.com/developerworks/library/wsbpel/.

Web Service Modelling Ontology, (2004). http://www.wsmo.org.

R. J. Wieringa (2003). Design Methods for Reactive Systems. MorganKaufmann: Elsevier Science, San Francisco.

Qiwen Xu and Mohalik Swarup, (1998). Compositional reasoning using the assumption-commitment paradigm. Lecture Notes in Computer Science, 1536:565–583.

Cau A. Xu Q. W. and Collette P (1994). On unifying assumptioncommitment style proof rules for concurrency. In B. Jonsson and Eds. J. Parrow, editors, In CONCUR'94, LNCS 836.

Shikun Zhou (2003). Compositional Framework for the Guided Evolution of TimeCritical Systems. PhD thesis, Software Technology Research Laboratory, De Montfort University UK.

J. Zwiers (1989). Compositionality, concurrency and partial correctness. SpringerVerlag New York, Inc., New York, NY, USA.