SEMANTICALLY ANNOTATING REACTIVE SERVICES WITH TEMPORAL SPECIFICATIONS

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

Abstract Most useful Web services are reactive systems that repeatedly act and react in interaction with their environment without necessarily terminating. Current Standards (XML based/Ontologies) for specifying behavioural semantics of services consider transformational aspects of service behaviour. Specification of reactive Web services require representation of properties that are temporal in nature. The properties need to be specified in a declarative form that is consistent with the dialect used for describing other aspects of the service. In the context of describing semantic Web services, these properties need to be expressed as ontologies that can be integrated with ontological representation frameworks for Web services for e.g. OWL-S. In this paper we present "TeSCO-S", a framework for enriching Web service interface specifications, described as OWL ontologies with temporal assertions. The TeSCO-S model is based on Interval Temporal Logic (ITL), our underlying formalism for reasoning about service behaviour over periods of time. 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 expressed in "Tempura", an executable subset of ITL.

Keywords: TeSCO-S, Interval Temporal Logic, Reactive, AnaTempura, Semantic Web Services, OWL

1. Introduction

Traditionally, Web services have been thought of and designed as being information-intensive, transformational systems. In order to fully utilise the distributed nature of control provided by interactive services deployed on the web, a change in the paradigm is desired. Web services may provide and transform information, but they may also exert control over their environment, facilitate behaviour, prevent behaviour and facilitate communication. 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 fig. 1. The service may also exert control over the environment by terminating the user session after pre-specified time limits of inactive sessions. Further, once a flight has been booked, the agent also has the option of cancelling the booking within a stipulated time period. From this and several related scenarios, we observe that most useful Web services repeatedly act and react in interaction with their environment without necessarily terminating. Their behaviour can be considered analogous to an important class of systems called "Reactive systems" [1] [14, 15, 23, 8]. Reactive Web services and their compositions generate complex

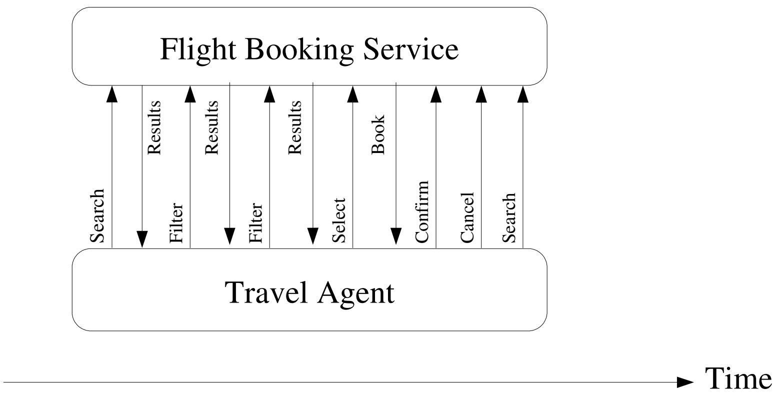


Figure 1. A Typical Flight Reservation Scenario

behaviours' due to their continuous interaction with the environment. Their execution may last anywhere from a few minutes to a few months. Examples include, web services deployed and composed as e-commerce applications, where an order once placed may 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-businesses, 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

[1]Reactive systems can be contrasted with transformational systems that take inputs, once in the beginning and produce outputs, once before terminating

services deployed on wireless devices may take more than expected time to provide the requested service due to poor connection facilities. In general, reactive services are required to satisfy real time constraints in response to the behaviour of the environment. For e.g. the database of the flight reservation service must be updated fast enough for all requests to be considered in a timely manner. Reactivity emerges when services communicate to form networks or compositions. For a service executing as part of a network, the environment is composed of all other services executing in that network. Further, Web services are perceived as black boxes where the internal computation of a process is not known. Reactive behaviour therefore needs to be specified, both at the abstract and declarative level as part of the service description.

An important aspect of reactive Web service specification is the representation of properties/rules that enable/constrain the behaviour of services and their continuous interactions with other services i.e. properties that are temporal in nature. High level description of services need to be enriched with properties which would enable reasoning about "ongoing" service behaviour. The properties need to be specified in a declarative form that is consistent with the language used for describing other aspects of the service. In the context of describing semantic Web services, these properties need to be expressed as ontologies that can be integrated with ontological representation frameworks for Web services. We observe that current Web service description frameworks suffer from the lack of their ability to fully specify properties that enable reasoning about reactive service behaviour and proving their correctness.

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 [12] 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 [30] as in verification, but for a particular trace or execution of a service. As shown in our earlier work on service composition [17, 26], 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 paper we propose a methodology to 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) [19, 20, 18, 3], 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 [7] 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.

The paper is organised as follows: We begin by presenting a motivating example of an e-Bookshop in section 2, which we follow throughout the paper, to explain vital concepts and constructs. Section 3 briefly discusses ITL, as our formal model for TeSCO-S. Section 4 provides a detailed presentation on TeSCO-S and its architecture. Section 5 discusses the relationship of TeSCO-S with existing standards and finally section 6 outlines conclusions and ongoing work.

2. A Motivating Example

The e-Bookshop as shown in fig 2 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. 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.

- 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 regis-

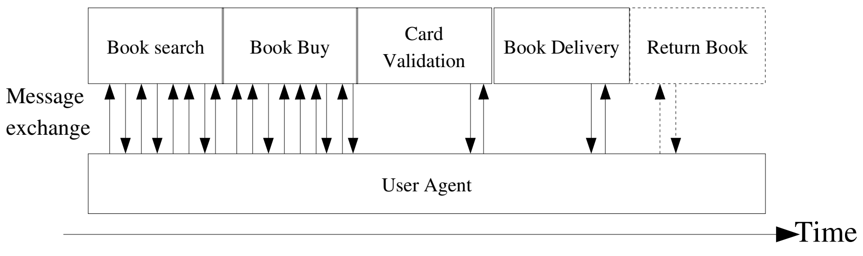


Figure 2. A Typical Book Buying Scenario

tration 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.
- 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.

3. Interval Temporal Logic

In this section we present the formal framework underlying TeSCO-S. We base our work on Interval Temporal Logic (ITL) and its executable subset Tempura. Our selection of ITL is based on a number of points. It is a flexible notation for both propositional and first-order reasoning about periods of time. Unlike most temporal logics, ITL can handle both sequential and parallel composition and offers powerful and extensible specification and proof techniques for reasoning about properties involving safety, liveness and projected time. Timing constraints are expressible and furthermore most imperative programming constructs can be viewed as formulas. Tempura, an executable subset of

ITL, provides a framework for developing, analysing and experimenting with suitable ITL specifications.

An interval is considered to be a (in)finite sequence of states, where a state is a mapping from variables to their values. An interval σ in general has a length $|\sigma| \geq 0$ and a nonempty sequence of $|\sigma| + 1$ states $\sigma_0 \ldots \sigma_{|\sigma|}$. Thus the smallest intervals have length 0 i.e. one state.

The syntax of ITL is defined below where μ is an integer value, a is a static variable (doesn't change within an interval), A is a state variable (can change within an interval), v a static or state variable, g is a function symbol and p is a predicate symbol. ITL contains conventional propositional operators such as

	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 3. Syntax of ITL

$\wedge$, $\neg$ 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 $\bigcirc$(next) and $\Box$(always). Expressions and formulae are evaluated relative to the beginning of an interval.

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). $skip$: $\overset{\sigma_0}{\bullet} \quad \overset{\sigma_1}{\bullet}$

- $f_1; f_2$: A formula $f_1; f_2$ is true on an interval σ with states $\sigma_0 \ldots \sigma_{|\sigma|}$ 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 $\sigma_0 \ldots \sigma_k$ and the subformula f_2 is true on the right part $\sigma_k \ldots \sigma_{|\sigma|}$.

- 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 4 pictorially represents the semantics of *skip*, *chop* and *chopstar*. Some ITL formula together with intervals which satisfy them are shown in fig 5

Some of the frequently used abbreviations are listed in Table 5. We do not present the formal semantics of ITL, due to lack of space. We refer the interested reader to [19, 20, 18, 3].

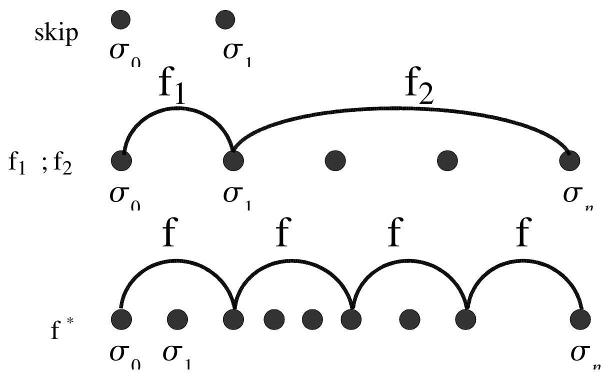


Figure 4. Informal illustration of ITL semantics

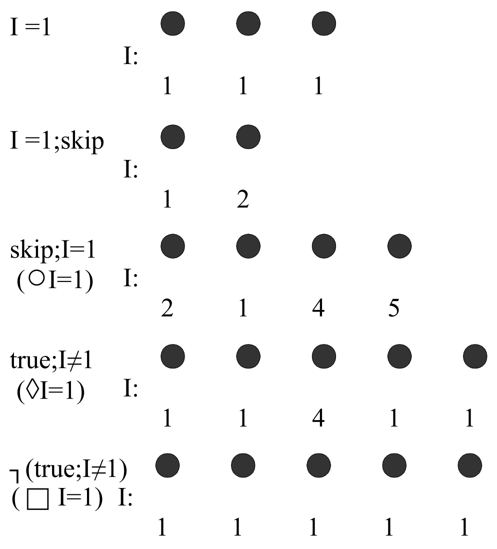


Figure 5. Some sample ITL formulae and satisfying intervals

3.1 Formalising the e-Bookshop in ITL

We now formalise some of the interesting properties of the e-Bookshop service from section 2. The temporal properties that are required to hold at various

$\bigcirc f$	$\widehat{=}$ skip ; f	next
more	$\widehat{=} \bigcirc$*true*	non-empty interval
empty	$\widehat{=} \neg$*more*	empty interval
$\Diamond f$	$\widehat{=}$ *finite* ; f	sometimes
$\Box f$	$\widehat{=} \neg\Diamond\neg f$	always
fin f	$\widehat{=} \Box(\text{empty} \supset f)$	final state
$\Diamond_i f$	$\widehat{=} f$; *true*	some initial subinterval
$\boxed{i} f$	$\widehat{=} \neg(\Diamond_i \neg f)$	all initial subintervals
$\Diamond_a f$	$\widehat{=}$ *finite* ; f ; *true*	some subinterval
$\boxed{a} f$	$\widehat{=} \neg(\Diamond_a \neg f)$	all subintervals

Table 1. Frequently used abbreviations

stages of the composition are as shown in fig 6. We also define the interval over which the properties are required to hold.

- 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.

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

- 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

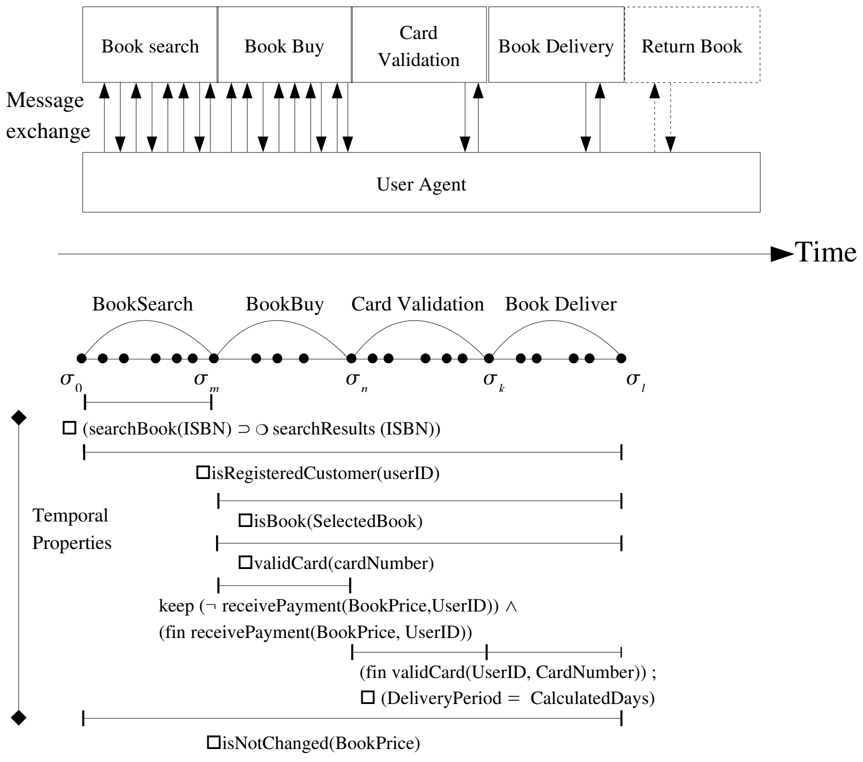

Book search
Book Buy
Card Validation
Book Delivery
Return Book
Message exchange
User Agent
Time
BookSearch
BookBuy
Card Validation
Book Deliver
σ0
σm
σn
σk
σl
□ (searchBook(ISBN) ⊃ ○ searchResults (ISBN))
□isRegisteredCustomer(userID)
□isBook(SelectedBook)
□validCard(cardNumber)
keep (¬ receivePayment(BookPrice,UserID)) ∧
(fin receivePayment(BookPrice, UserID))
(fin validCard(UserID, CardNumber)) ;
□ (DeliveryPeriod = CalculatedDays)
□isNotChanged(BookPrice)
Temporal Properties

Figure 6. Temporal Properties for the e-Bookshop

$(\sigma_m \dots \sigma_l)$.

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

- 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 $(\sigma_n \dots \sigma_l)$.

  $$(fin\ validCard(UserID, CardNumber));(DeliveryPeriod = CalculatedDays)$$

3.2 Assumption-Commitment properties for Web services

An important class of tempoal properties for Web services are “Assumption-commitment” properties. In our earlier work [26, 16, 17] on service composition, we have shown the power of assumption-commitment style of specification for compositional reasoning of ongoing service behaviour. In brief, assumption-commitment is a compositional specification and verification methodology for the precise and clear specification of the behaviour of reactive services. 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.

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.

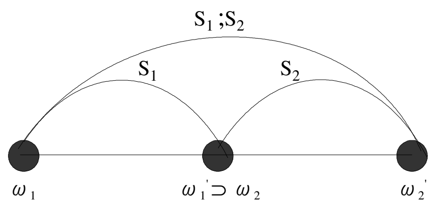


Figure 7. Sequential Composition

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
$\omega^{'}$	: state formula about final state

We consider the sequential composition (ref. Fig. 7) 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 [26, 17].

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

4. TeSCO-S: Temporal SemantiCs for OWL enabled Services

TeSCO-S is a framework (ref. Fig.8) 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.

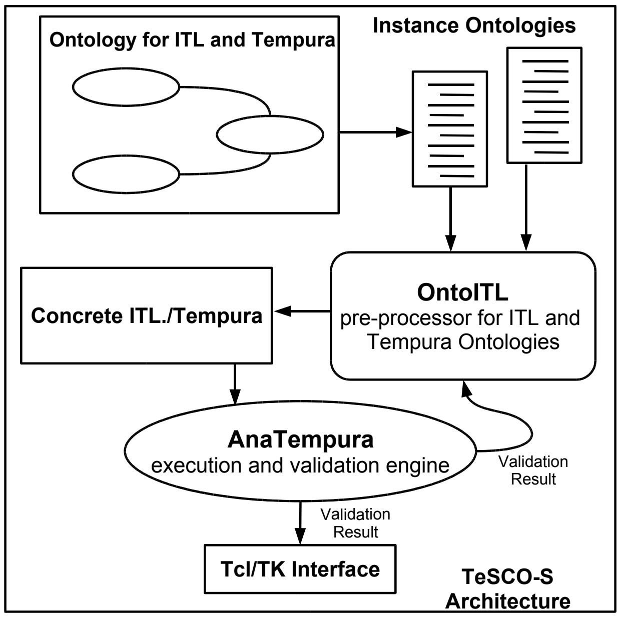


Figure 8. The TeSCO-S Architecture

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

4.1 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. "skip",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 4.2. The modular approach to building ITL and Tempura formulae allows reusability of formulae and expression instances between ontologies.

We use the Protege OWL plugin [4] for modelling the ontology. Table 4.1 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 [1]. The complete ontology itself can be found at [2].

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

Table 2. Primitives for the ITL-Tempura Ontology

4.2 Modelling the e-Bookshop service

In this section, we model some interesting properties of the e-Bookshop service 3.1 using the ITL-Tempua ontology. Since most of the properties are composite formulae, we begin by defining the "Composite" formula class in the abstract Description Logic (TBox) syntax [2].

[2]For brevity and readability purposes, we have abstained from providing the actual ontological representation. The syntax has been specified here as abstract Description Logic syntax. For more details, please refer to the ontology

Composite ⊑ Formula ⊓ (∀ hasPrefixedSubFormula.Formula)
⊓ (∀ hasSuffixedSubFormula.Formula) ⊓ (=1 hasInfixOperator.Operator)
⊓ (=1 hasPrefixedSubFormula.Formula) ⊓ (=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.

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

  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.

  $$(fin\ validCard(UserID, CardNumber));(DeliveryPeriod = CalculatedDays)$$

  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

4.3 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.

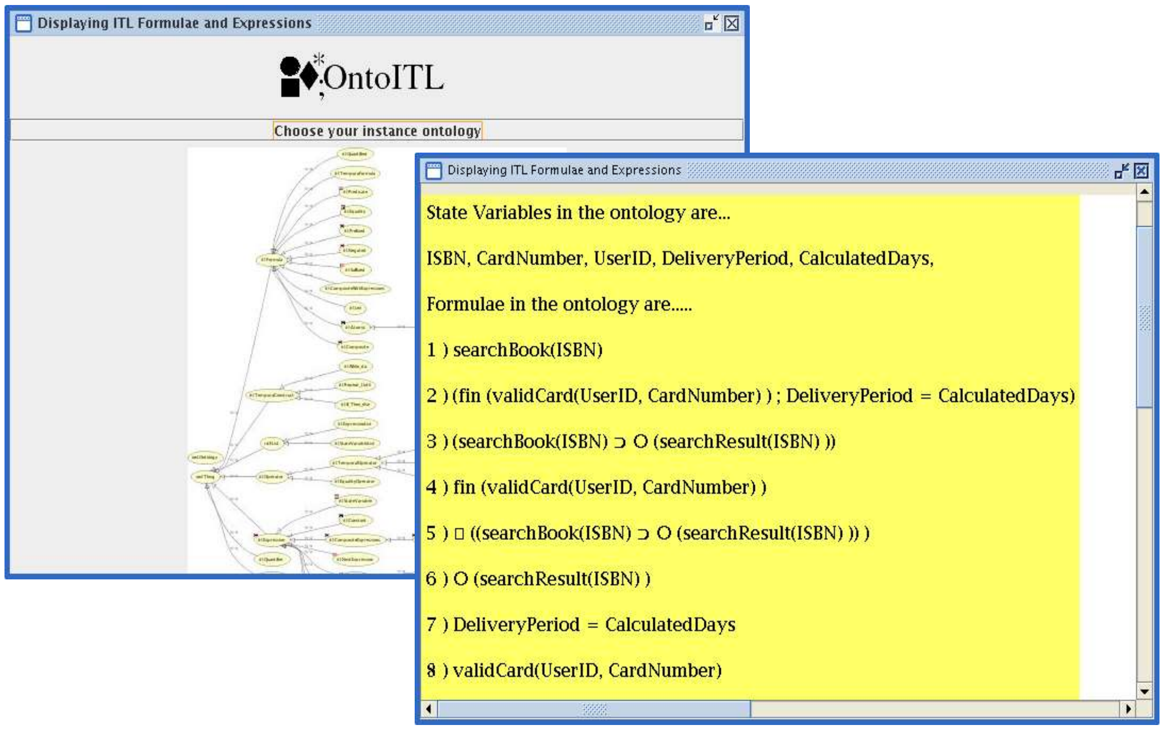
OntoITL is a pre-processor that generates concrete ITL and executable Tempura formulae from instance ontologies built using the ITL-Tempura Ontology. The instances are defined using the core ontology as described in Section 4.1 or from ontologies that import these instances. It provides as output, complete 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 4.2 is shown in the Fig. 9: 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 4.4.

4.4 AnaTempura: Runtime Validation of Tempura specification

AnaTempura (available from [3]), 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 Fig. 10. There are two ways of validating properties via AnaTempura:


Displaying ITL Formulae and Expressions
OntoITL
Choose your instance ontology
Displaying ITL Formulae and Expressions

State Variables in the ontology are...

ISBN, CardNumber, UserID, DeliveryPeriod, CalculatedDays,

Formulae in the ontology are.....

1) searchBook(ISBN)

2) (fin (validCard(UserID, CardNumber)) ; DeliveryPeriod = CalculatedDays)

3) (searchBook(ISBN) ⊃ ○ (searchResult(ISBN)))

4) fin (validCard(UserID, CardNumber))

5) □ ((searchBook(ISBN) ⊃ ○ (searchResult(ISBN))))

6) ○ (searchResult(ISBN))

7) DeliveryPeriod = CalculatedDays

8) validCard(UserID, CardNumber)

Figure 9. The OntoITL pre-processor for ITL-Tempura Ontology

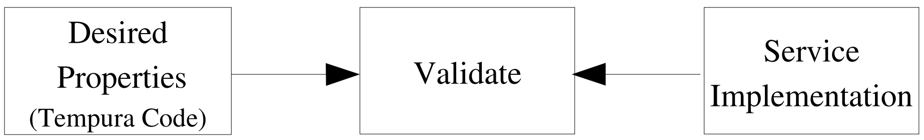


Figure 10. The Analysis Process

- 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.

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 Fig. 11. The validation results of the instance-

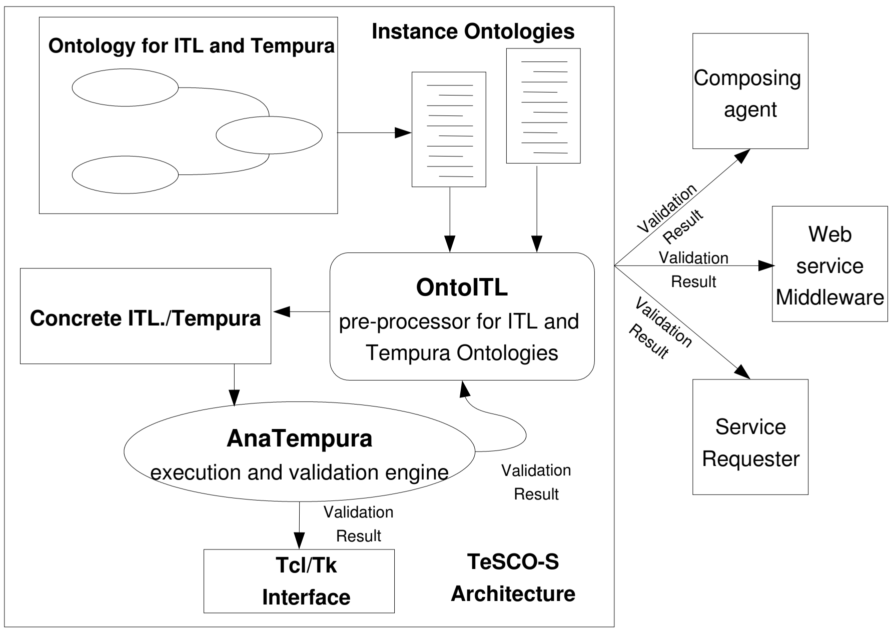


Figure 11. General Architecture for Web services

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.

4.5 Validating the Customer : e-Bookshop Composition

We have validated some of the properties of the e-Bookshop as formalised in section 3.1. We present the validation of one such property,

$$\Box(isRegistered(UserID))$$

We adopt the second approach to validating properties as mentioned in section 4.4. 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 [3]. The login details are set for the customer session and passed to AnaTempura. For each

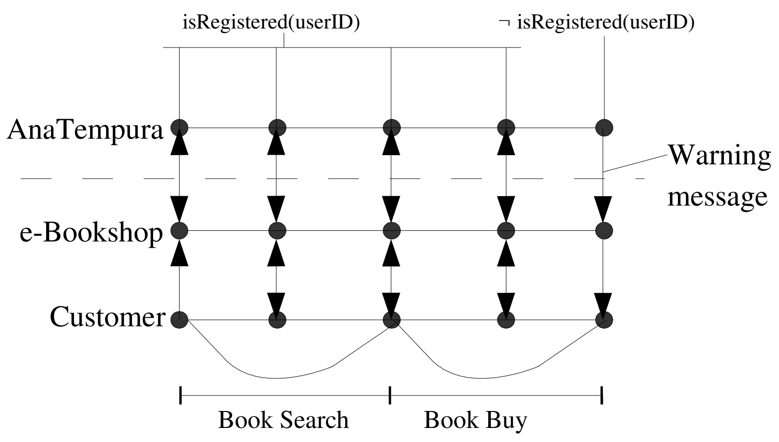


Figure 12. Validating the Customer:e-Bookshop Composition

phase of the composition (search, buy etc.) and for every interaction between the e-Bookshop and the customer, at any state, the property is validated by AnaTempura against the values set in the session for that state. If the values in the session are found to be reset and do not match the ones passed to AnaTempura in the initial state, a warning message is sent to the e-Bookshop as shown in fig. 12. It is worth noting that AnaTempura only validates the properties of interest. It does not define the behaviour of the service in case the properties are not satisfied. This is a design decision that has to be taken before the composition is realised.

[3]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.

5. Relationship with Existing standards

Based on service interfaces definitions [24] and message exchange protocols [5], standards [10, 28, 21, 25, 27] have been proposed for specifying composite services, by defining declaratively, their data and control flows. BPEL4WS [28] 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 [27] and WSMO [29], 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 modeling of "states" in these languages. Rule languages for the web include RuleML [6] and within the context of semantic web, initiatives such as SWRL [13] and DRS [11]. 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 [22, 9] i.e. reasoning about time points, complex time intervals, calendars and durations.

TeSCO-S provides an OWL ontology for modelling temporal properties of services and a tool to validate them. In our earlier work, we have shown how temporal properties for services can be declaratively specified in SWRL. The ITL-Tempura ontology provides richer expressiveness in the specification of properties due to more concepts and properties being available. It can be integrated very easily with SWRL and with existing Web service standards that describe service interfaces in OWL. As an example, the definition of "State-Variable " can be extended to define it as a SWRL variable. Similarly "head" and "body" atoms can be defined in terms of "Predicate" in the ITL ontology. Temporal properties that define execution monitoring of OWL-S services can be readily defined as Tempura formulae, which AnaTempura validates at runtime.

6. Conclusion and Future Work

In this paper, 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 preconditions/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.

References

[1] A Graphical representation of Class Hierarchies in the ITL-Tempura Ontology. `http://www.cse.dmu.ac.uk/~monika/TeSCO-S/OntoITL.jpg`.

[2] An Ontology for ITL and Tempura. `http://www.cse.dmu.ac.uk/~monika/TeSCO-S/OntoITL.owl`.

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

[4] The protege ontology editor and knowledge acquisition system.

[5] Simple Object Access Protocol. http://www.w3.org/TR/2002/CR-soap12-part0-20021219/.

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

[7] OWL Web Ontology Language Reference, 10 February 2004. `http://www.w3.org/TR/owl-ref/`.

[8] A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems - a survey of current trends. *Current trends in Concurrency*, LNCS 224:510–584, 1986.

[9] F. Bry and S. Spranger. Temporal constructs for a web language, 2003.

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

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

[12] E.M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* The MIT Press, Cambridge, Massachusetts, 1999.

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

[14] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, New York, 1991.

[15] Z. Manna and A. Pnueli. *The Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, New York, 1995.

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

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

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

[19] 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.

[20] 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.

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

[22] Feng Pan and Jerry R. Hobbs. Time in OWL-S. In *Proceedings of AAAI Spring Symposium Series on Semantic Web Services*, 2004.

[23] A Pnueli. Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends. pages 510–584, 1986.

[24] 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.

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

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

[27] The OWL-S Coalition. OWL-S 1.1 Release., 2004. `http://www.daml.org/services/owl-s/1.0/`.

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

[29] Web Service Modelling Ontology, 2004. http://www.wsmo.org.

[30] Shikun Zhou. *Compositional Framework for the Guided Evolution of Time-Critical Systems*. PhD thesis, Software Technology Research Laboratory, De Montfort University UK, 2003.