An Integrated Formal Approach to Semantic Work Environments Design

Hai H. Wang, University of Southampton, hw@ecs.soton.ac.uk
Jin Song Dong, National University of Singapore, dongjs@comp.nus.edu.sg
Jing Sun, The University of Auckland, j.sun@cs.auckland.ac.uk
Terry R. Payne, University of Southampton, trp@ecs.soton.ac.uk
Nicholas Gibbins, University of Southampton, nmg@ecs.soton.ac.uk
Yuan Fang Li, National University of Singapore, liyf@comp.nus.edu.sg
Jeff Pan, University of Aberdeen, jpan@csd.abdn.ac.uk

Keywords: *Formal Methods, Semantic Web Services, Ontology, OWL, OWL-S*

1. Introduction

The Semantic Web (Berners-Lee, 2001) has become increasingly significant as it proposes an evolution of the current World Wide Web, from a web of documents to a distributed and de-centralised, global knowledge-base. Based on the notion of interlinked resources grounded within formally defined ontologies, it promises to be an enabling technology for the automation of many Web-based tasks, by facilitating a shared understanding of a domain through inference over shared knowledge models. Semantic Work Environment (SWE) applications use Semantic Web techniques to support the work of the user by collecting knowledge about the current needs in a specific activity, and providing both inferred and augmented knowledge that can then be integrated into current work. Web Services have emerged as distributed, heterogeneous software components that provide machine access to the services otherwise offered on the web through Web Pages. Built upon de-facto Web standards for syntax, communication protocols, and markup languages such as XML, Web Services provide a near ubiquitous mechanism for communication between applications and agents. In addition, such services can be composed to provide additional functionality, thus facilitating the rapid construction of new services. However, the dynamic use of services is limited by the need to agree a-priori upon data models and interface definitions. By coupling Web Service technology with Semantic Web technology, *Semantic Web Services* can partially relax these constraints, both in the dynamic use of services, and in the data models shared by such services. Several examples of such services have been developed; for example, the ITTALKS services (Cost, 2002), which are considered in this chapter.

The use of reasoning over shared domain models, and the description of services and capabilities using Semantic Web Service descriptions is essential in supporting communication and collaboration between agents. Agents may agree on the same vocabulary and domain for communication, but different agents may not necessarily refer to the same concept in the same way. The Semantic Web is by definition highly distributed, and is built upon the assumption that different parties may have a different conceptualisation or representation for the same concept. Agreement on the concepts and the terms used to reference them, their relationships with other concepts, and the underlying aximoatisation of a domain model is addressed by formally defining shared ontologies. These ontologies represent different domains, and through inference, can identify the equivalence of two concepts that may have different representations. The OWL Web Ontology Language (Dean, 2004) is a W3C recommendation for representing ontologies based on Description Logics, and provides a basis for providing both terminologies (i.e. vocabularies, relationships, axioms and rules) and knowledge bases. Whilst providing the mechanisms for defining domain ontologies, representational ontologies (Heijst, 1997) are necessary for defining the structure of services themselves, and several such ontologies have been proposed (Ankolekar, 2002; Roman, 2005). These ontologies define at a meta-level what the service does, how to use it, what are its outputs and effects, etc, and thus can facilitate the run-time use of services (including discovery and execution) and consequently problem solving without necessitating human intervention at that point. OWL-S (Ankolekar, 2002) is one such ontology (defined in OWL-DL) that describes Web Services, including models for service orchestration (through the process model) and mappings to Web Service definitions.

Services found in Semantic Work Environments (SWE) can be defined using Semantic Web Service ontologies, and grounded within a specific domain by using the relevant, shared domain ontologies. The services may have intricate data states, complex process behaviours and concurrent interactions. The design of such systems requires precise and powerful modelling techniques to capture not only the ontology domain properties but also the services' process behaviours and functionalities. It is therefore desirable to have a powerful formal notation that captures these notions, in addition to the declarative ontological representations of the domains and the services themselves, to precisely design Semantic Work Environments.

Timed Communicating Object-Z (TCOZ) (Mahony, 2000) is a Formal Specification language which builds on the strengths of Object-Z (Duke, 2000; Smith, 2000) in modelling complex data and state with the advantage of Timed CSP (Schneider, 1995) in modelling real-time concurrency.
The following characteristics of many SWE applications make TCOZ a good candidate to design such a system:

- A complex SWE applications system often has both intricate data state and process control aspects. An integrated formal modeling language, like TCOZ, has the capability to model such systems.
- A service-providing agent may offer several kinds of different services concurrently. TCOZ can support this through its multithreaded capabilities.
- A complex SWE system is often composed from many individual services. These other services may be provided by other agents, which have their own threads of control. This can be modeled by the active-objects feature in TCOZ.
- A SWE application may include highly distributed components with various synchronous and asynchronous communication channels. This can be specified with various TCOZ communication interfaces: namely *channels*, *sensors* and *actuators*.
- Some SWE applications, such as online hospital or online bank applications may have critical timing requirements. These real-time requirements can also be captured by TCOZ.

Thus, we propose to use TCOZ as a language to model the complex SWE applications. We believe one effective way to design a complex system is to use a single expressive modeling language, like TCOZ, to present complete, rigorous and coherent requirements models for complex system as a specification contract. Later on, this complex model can be projected into multiple domains so that existing specialized tools in these corresponding domains can be utilized to perform the checking, analysis and other tasks. We have developed tools to automatically map TCOZ to *UML* for visualization purpose (Dong, 2002c), to *Timed Automata* for checking the time-related properties, to OZ for animation (Sun, 2003) and to *Isabelle/HOL* for complex reasoning (Sun, 2003) etc.

We believe that TCOZ, as a high-level design technique can contribute to the SWE application development in many ways. In support of this claim, we have conducted a semantic web service case study, i.e., the online talk discovery system, and apply TCOZ to the design stage to demonstrate how TCOZ can be used to augment and model semantic web services. Using an expressive formal language like TCOZ can provide an unambiguous requirement for the SW service system and the series of related supporting tools (Dong, 2002c; Sun, 2001a; Sun, 2003) can ensure high quality from the design model. In addition to presenting these general advantages of using formal language for designing systems, the chapter also presents the development of a set of systematic translation rules and tools for automatically extracting the Web ontology and generating the resulting semantic markup for the SW services from the formal TCOZ design model. It is a desired add-on value, as designing the Web ontology and semantic markup for the SW services itself is a tough task for the domain engineers. Those transformation rules are non-trivial. For example, the semantics of the OWL subclass construct is different to that of the Z schema inclusion construct, or TCOZ class inheritance, and such differences should be managed appropriately. Rigorous study has been made to avoid the conflict between those semantics. Our online talk discovery system is a simplified variant of the ITTALKS system (Cost, 2002), which is a deployed semantic web service suite that has been used extensively.

The remainder of the chapter is organized as follows. Section 2 briefly introduces TCOZ and SW. Section 3 formally specifies the functionalities of the SW service example (i.e. the talk discovery system). Section 4 presents the tool that extracts the ontology used by the SW services from the TCOZ design model automatically. Section 5 presents the tool which extracts the semantic markup for SW services from the TCOZ design model automatically. Finally, the chapter concludes in Section 6.

2. TCOZ and SW Services overview

TCOZ overview

Object-Z and CSP

Object-Z (Duke, 2000) is an extension of the Z formal specification language to accommodate object orientation. The main reason for this extension is to improve the clarity of large specifications through enhanced structuring. Although Object-Z has a type checker, other tool support for Object-Z is somewhat limited in comparison to Z. The essential extension to Z in Object-Z is the *class* construct, which groups the definition of a state schema with the definitions of its associated operations. A class is a template for **objects** of that class: for each such object, its states are instances of the state schema of the class and its individual state transitions conform to individual operations of the class. An object is said to be an instance of a class and to evolve according to the definitions of its class.
Timed CSP (TCSP) (Schneider, 1995) extends the well-known *CSP* (Communicating Sequential Processes) notation with timing primitives. As indicated by its name, CSP is an *event*-based notation primarily aimed at describing the sequencing of behaviour within a process and the synchronization of behaviour (or *communication*) between processes. Timed CSP extends CSP by introducing a capability to consider the temporal aspects of sequencing and synchronization.

CSP adopts a symmetric view of both process and environment, with events representing a co-operative synchronization between them. Both process and environment may control their behaviour by *enabling* or *refusing* certain events or sequences of events. The primary building blocks for Timed CSP processes are *sequencing, parallel composition,* and *choice*.

TCOZ features

Timed Communicating Object-Z (TCOZ) (Mohony, 2000) is essentially a blending of Object-Z with Timed CSP (Schneider, 1995), for the most part preserving them as proper sub-languages of the blended notation. The essence of this blending is the identification of Object-Z operation specification schemas with terminating CSP processes. Thus, operation schemas and CSP processes occupy the same syntactic and semantic category; operation schema expressions may appear wherever processes may appear in CSP, and CSP process definitions may appear wherever operation definitions may appear in Object-Z. The primary specification structuring device in TCOZ is the Object-Z class mechanism.

Below, we briefly consider the various aspects of TCOZ. A detailed introduction to TCOZ and its Timed CSP and Object-Z features may be found elsewhere (Mahony, 2000), where the formal semantics of TCOZ are also documented.

Interface -- channels, sensors and actuators

CSP channels are given an independent, first class role in TCOZ. In order to support the role of CSP channels, the state schema convention is extended to allow the declaration of communication channels. If *c* is to be used as a communication channel by any of the operations of a class, it must be declared in the state schema to be of type *chan*. Channels are type heterogeneous and may carry communications of any type. Contrary to the conventions adopted for internal state attributes, channels are viewed as shared (i.e. global) rather than as encapsulated entities. This is an essential consequence of their role as communications interfaces *between* objects. Thus, the introduction of channels to TCOZ reduces the need to reference other classes in class definitions, thereby enhancing the modularity of system specifications.

Complementary to the synchronizing CSP channel mechanism, TCOZ also adopts a non-synchronizing shared variable mechanism. A declaration of the form *'s: X sensor'* provides a channel-like interface for using the shared variable *s* as an input. A declaration of the form *'s: X actuator'* provides a local-variable-like interface for using the shared variable *s* as an output. Sensors and actuators may appear either at the system boundary (usually describing how global analogue quantities are sampled from, or generated by the digital subsystem) or else within the system (providing a convenient mechanism for describing local communications which do not require synchronization)[1]. The shift from closed to open systems necessitates close attention to issues of control, an area where both Z and CSP are weak (Zave, 1997).

Active objects

Active objects have their own threads of control, while passive objects are controlled by other objects in a system. In TCOZ, an identifier *MAIN* (non-terminating process) is used to determine the behaviours of active objects of a given class (Dong, 1998). The *MAIN* operation is optional in a class definition. It only appears in a class definition where the objects of that class are active objects. Classes for defining passive objects will not have the *MAIN* definition, but may contain CSP process constructors. If *ob1* and *ob2* are active objects of the class *C*, then the independent parallel composition behaviours of the two objects can be represented as $ob1 \interleave ob2$, which means $ob1.MAIN \interleave ob2.\ MAIN$.

The syntactic implication of the above approach is that the basic structure of a TCOZ document is the same as for Object-Z. A document consists of a sequence of definitions, including type and constant definitions in the usual Z style. TCOZ varies from Object-Z in the structure of class definitions, which may include CSP channel and processes definitions. Let us use a simple timed message queue system to illustrate the TCOZ notation. The behaviour of the following timed message queue system is that it can receive a new message (of type [MSG]) through an input channel *in* within time duration 'T_j' or remove a message and send it through an output channel 'out' within time duration 'T_1'. If there is no interaction with environment within a certain time 'T_o', then a message will be removed from the current list

[1] Mahony presented detailed discussion on TCOZ sensor and actuators in (Mahony and Dong, 1999).

but stored in a (window like) actuator list (*lost*) so that other objects (un-specified) with a sensor ‘lost’ can read it at any time. The message queue has a FIFO property.

$TimedQueue$

$$\begin{array}{l} items : \mathrm{seq}\, MSG \\ in, out : \mathbf{chan} \\ lost : \mathrm{seq}\, MSG\ \mathbf{actuator} \\ T_l, T_j, T_o : \mathbb{N} \end{array}$$

$\textsc{Init}$

$$items = lost = \langle\ \rangle$$

$RecLost$

$$\begin{array}{l} \Delta(lost) \\ \hline lost' = \langle head(items) \rangle ^\frown lost \end{array}$$

Add

$$\begin{array}{l} \Delta(items) \\ i? : MSG \\ \hline items' = items ^\frown \langle i? \rangle \end{array}$$

Del

$$\begin{array}{l} \Delta(items) \\ i! : MSG \\ \hline items \neq \langle\ \rangle \Rightarrow items = \langle i! \rangle ^\frown items' \\ items = \langle\ \rangle \Rightarrow items' = \langle\ \rangle \end{array}$$

$$\begin{array}{l} Join \mathrel{\widehat=} [i : MSG] \bullet in?i \rightarrow Add \bullet \textsc{Deadline}\ T_j \\ Leave \mathrel{\widehat=} [items \neq \langle\ \rangle] \bullet out!head(items) \rightarrow Del \bullet \textsc{Deadline}\ T_l \\ \textsc{Main} \mathrel{\widehat=} \mu\, Q \bullet (Join \,\Box\, Leave) \rhd\{T_o\}\ (RecLost;\ Del) \bullet \textsc{Deadline}\ T_l;\ Q \end{array}$$

As we can see, Object-Z and TCSP complement each other not only in their expressive capabilities, but also in their underlying semantics. Object-Z is an excellent notation for modelling data and states, but difficult for modelling real-time and concurrency. TCSP is good for specifying timed process and communication, but like CSP, it can be cumbersome when capturing the data states of a complex system. By combining the two, TCOZ treats data and algorithmic aspects using Object-Z, whilst treating process control, timing, and communication aspects using TCSP constructs. In addition, the object-oriented flavour of TCOZ provides an ideal foundation for promoting modularity and separation of concerns in system design. With the above modelling abilities, TCOZ is potentially a good candidate for specifying composite systems in a highly constructive manner.

Semantic Web Ontology and Service overview

As a huge, distributed, information space, the World Wide Web was originally designed to seamlessly support human navigation through related, linked documents. Although this medium was originally designed to do more than simply support human-to-human communication (Berners-Lee, 2001), machine or agent mediated assistance has been hindered by the type of markup used within the documents. An emphasis by content providers on presentation and physical design has resulted in a lack of structure, both at the layout and content levels, and rendered most documents opaque to machine comprehension. Automated approaches to extract knowledge or data from such web pages, or to simulate human browsing and activity with interactive pages required the a priori construction of messages (using http) to web servers, and the subsequent parsing of HTML to extract the desired data. These tasks were achieved either manually by software engineers, or through identifying regularities through the use of machine learning techniques (Lieberman, 2001) within the raw HTML. Such approaches were fragile, and could easily fail if the message format was changed (as was frequently the case) or if new or unexpected content was returned in the resulting HTML. The emergence of XML and XHTML has been significant in addressing the problem of a lack of structure within web pages, and the evolution of Web Services as a near-ubiquitous, standard technology has greatly facilitated the automated use of services on the web. However, although XML was designed to define the syntax of a document, it says nothing about the semantics of entities within a document, and consequently does not assist in the interpretation or comprehension of messages or exchange sequences (Bussler, 2001).

The Semantic Web provides a mechanism for including semantics into XML-based structured documents by building upon various notions underlying knowledge-based systems: namely the use of a modeling paradigm to model a domain; a representation language to support the sharing of facts (i.e. instances) and models (i.e. ontologies) between agents and applications, and reasoning mechanisms to facilitate the inference of any facts entailed by the model.

Ontologies are an explicit, formal specification of a shared conceptualisation of a domain (Studer, 1998): they provide a machine readable, and agreed upon representation of an abstraction of some phenomenon. This typically involves defining the concepts within the domain being modeled, their properties and relationships with other concepts. In some cases, ontologies may be complemented by axioms, statements that are always true and that are used to constrain the meaning of concept definitions in the ontologies. Often the declarative definitions are not sufficient to constrain completely the meaning of concepts and to capture the “procedural” or decision making aspects of the application business logic. Therefore ontologies may need to be complemented by rules, grounded in ontological definitions, to facilitate enhanced representation and reasoning capabilities.

Whilst a variety of knowledge representations have been proposed in the past, basing a representation on an XML syntax facilitates the use of Web-based machinery for a variety of tasks, including indexing statements, concepts and properties (URIs), defining vocabularies (RDF Schema) (Brickley, 2004) and representing statements about objects (through RDF’s subject-predicate- object model). The Web Ontology Language, OWL is an XML-based language for representing Description Logic terminologies (i.e. vocabularies, relationships, axioms and rules) and knowledge bases. Whilst many logics may be expressionally rich, and can support sophisticated knowledge representations, such logics are mostly computationally intractable. A significant advantage of utilizing Description Logics (DLs) for expressing knowledge is that the tractability of different DL subsets (with varying expressivity) is better understood, and in some cases formally proved. Whilst limiting the expressivity of a language may reduce the scope of knowledge that can be represented, ontologies have been developed and successfully deployed for pragmatic use in domains such as medical research and eScience. Several highly-optimised reasoning mechanisms have also been developed that support variants of DLs that whilst theoretically are intractable, are pragmatically tractable for most real-world problems.

Three increasingly expressive sublanguages of OWL have been defined to date: OWL-Lite, OWL-DL and OWL-Full. OWL-Lite offers limited expressivity in order to facilitate rapid uptake, and to support the definition of simple, taxonomic structures with limited axioms; whereas OWL-DL has been designed with the intent to support DL representations of the application business logic, and to provide a language subset that has desirable computational properties for reasoning systems. Finally, OWL-Full encompasses maximal expressivity, and as a result is not decidable, and makes no computational guarantees. The entailed knowledge represented within such ontologies may be inferred through a variety of different reasoning mechanisms.

Semantic Web Services provide a declarative, ontological framework for describing services, messages, and concepts in a machine-readable format that can also facilitate logical reasoning. Thus, service descriptions can be interpreted based on their meanings, rather than simply a symbolic representation. Provided that there is support for reasoning over a semantic web service description (i.e. the ontologies used to ground the service concepts are identified,

or if multiple ontologies are involved, then alignments between ontologies exist that facilitate the transformation of concepts from one ontology to the other), workflows and service compositions can be constructed based the semantic similarity of the concepts used.
OWL-S is an OWL-based Web service ontology, which supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-intepretable form. OWL-S was expected to enable the automatic Web service discovery, automatic Web service invocation and automatic Web service composition and interoperation. OWL-S consists of three essential types of knowledge about a service: the profile, the process model and the grounding. The OWL-S *profile* describes *what the service does*. The OWL-S *process model* tells *how the service works*. The OWL-S *grounding* tells *how the service is used*. The OWL-S process model is intended to provide a basis for specifying the behaviors of a wide array of services. There are two chief components of an OWL-S process model – the process, and process control model. The process describes a Web Service in terms of its input, output, precondition, and effects, where appropriate, its component subprocess. The process model enables planning, composition and agent/service inter-operation. The process control model -- which describes the control flow of a composite process and shows which of various inputs of the composite process are accepted by which of its subprocesses -- allows agents to monitor the execution of a service request. The constructs to specify the control flow within a process model includes *Sequence*, *Split*, *Split+Join*, *If-Then-Else*, *Repeat-While* and *Repeat-Until*.

3. The talk discovery system

In this section, an online talk discovery system is used as an example to demonstrate how TCOZ notation can be applied to the Semantic Web service development.

System scenario

The talk discovery system is a Web portal offering access to information about talks (i.e. presentations) and seminars. This Web portal can provide not only the talk's information corresponding to the user's profile in terms of his interest and location constraints, but also can further filter the related talks based on information about the user's personal schedule, etc.
In the course of operation, the talk discovery system discovers that there is an upcoming talk that may be of interest a registered user based on the information in the user's preferences, which have been obtained from his online, OWL-encoded profile. Upon receiving this information, the user's User Agent gathers further information: it consults with its Calendar agent to determine the user's availability, and with the MapQuest agent to find the distance from the user's office to the talk's venue. Finally, after evaluating the information and making the decision, the User Agent will send a notification back to the talk discovery agent indicating that the user will (or will not) plan to attend.

Formal model of the talk discovery system

The system involves four different intelligent agents which communicate interactively. They are the user's *Calendar agent*, the *MapQuest agent*, user's *Personal agent* and the *Talk Discovery agent*.

Calendar agent

Firstly, the *DATE* and *TIME* set are defined by the Z given type definitions. As this chapter focuses only on demonstrating the approach, the model we present here has been kept simple. Z given type is chosen to define different concepts, including *TIME* and *DATE*. These concepts can be subdivided into detailed components, e.g., the *TIME* comprises hour, minute, and second. The more detailed the model, the more detailed the ontology will be when derived automatically from our tool (this tool will be further discussed in the later sections).

The *DateTime* concept is defined as a schema with two attributes date and time.

$[TIME, DATE]$

$$\begin{array}{|l}\hline \textit{DateTime} \\ date : DATE \\ time : TIME \\ \hline\end{array}$$

The Calendar agent maintains a schedule for each eligible user and supplies related services. Each eligible user must have a personal ID *[PID]* registered. This *id* is used to validate the identity of users when the system receives requests. The Calendar agent has an *ID manager* which provides functions for certifying identity. It may use Web security techniques such as digital signatures, to ensure the service is only available to the valid users. The following schema specifies the ID manager. The attribute *ids* denotes the set of customs' ID registered to the system. The *PIDMananger* can receive a new *id* through an input channel *add* (*New*) and add it tot its database, remove an *id* from its database (*Delete*) or check if a custom has registered (*Validate*).

$$\begin{array}{|l}\hline \textit{PIDManager} \\
\begin{array}{|l}\hline ids : \mathbb{P}\, PID \\ add, remove : \mathbf{chan} \\ check : \mathbf{chan} \\ \hline\end{array}
\quad
\begin{array}{|l}\hline \textsc{Init} \\ ids = \varnothing \\ \hline\end{array} \\
\begin{array}{|l}\hline AddPID \\ \Delta(ids) \\ id? : PID \\ \hline ids' = ids \cup \{id?\} \\ \hline\end{array}
\quad
\begin{array}{|l}\hline RemovePID \\ \Delta(ids) \\ id? : PID \\ \hline ids' = ids - \{id?\} \\ \hline\end{array} \\
New \mathrel{\widehat=} [id : PID \mid id \notin ids] \bullet add?id \to AddPID \\
Delete \mathrel{\widehat=} [id : PID \mid id \in ids] \bullet remove?id \to RemovePID \\
Validate \mathrel{\widehat=} [id : PID] \bullet check?id \to \\
\qquad\qquad ([id \in ids] \bullet check!\mathrm{true} \to \textsc{Skip} \\
\qquad\qquad \Box\ [id \notin ids] \bullet check!\mathrm{false} \to \textsc{Skip}) \\
\textsc{Main} \mathrel{\widehat=} \mu\, N \bullet (New \ \Box\ Delete \ \Box\ Validate);\ N \\ \hline\end{array}$$

The *Status* ($Status ::= FREE \mid BUSY$) defined by the Z free type definition indicates if a person is free or busy. *Update,* defined in *Calendar,* is used to update the timetable and it must complete its task within 1 second. This real time property is captured by TCOZ's *DEADLINE* operator. The operation *Check_Status* is used to check whether a person is available or not for a particular time slot.

Calendar

$timetable : (PID \times DateTime) \rightarrow Status$
$upd, checktm : \mathbf{chan}$
$check : \mathbf{chan}$

Upd

$\Delta(timetable)$
$id? : PID;\ t? : DateTime;\ \ s? : Status$

$timetable' = timetable \oplus \{(id?, t?, s?)\}$

$$\begin{aligned}
Update \mathrel{\widehat=}\ & [id : PID;\ t : DateTime;\ s : Status] \bullet upd?(id, t, s) \\
& \rightarrow check!id \rightarrow (check?\,\mathrm{false} \rightarrow \textsc{Skip} \ \Box\ check?\,\mathrm{true} \rightarrow Upd) \\
& \bullet \textsc{Deadline}\ 1 \\
Check\_Status \mathrel{\widehat=}\ & [id : PID;\ t : DateTime] \bullet checktm?(id, t) \\
& \rightarrow check!id \rightarrow \\
& \quad (check?\,\mathrm{false} \rightarrow \textsc{Skip}\ \Box \\
& \quad check?\,\mathrm{true} \rightarrow checktm!timetable(id, t) \rightarrow \textsc{Skip}) \\
\textsc{Main} \mathrel{\widehat=}\ & \mu N \bullet (Update \ \Box\ Check\_Status);\ N
\end{aligned}$$

MapQuest agent

The MapQuest agent is a agent supplying the service for calculating the distance between two places. Firstly, *PLACE* is defined as a Z given type [PLACE]. The MapQuest agent contains a set of places in its domain and a database storing the distance between any two places.

MapQuest

$places : \mathbb{P}\, PLACE$
$distance : places \times places \rightarrow \mathbb{R}^+$
$dist : \mathbf{chan}$

$$\begin{aligned}
Get\_dis \mathrel{\widehat=}\ & [p_1, p_2 : places] \bullet dist?(p_1, p_2) \rightarrow \\
& dist!distance(p_1, p_2) \rightarrow \textsc{Skip} \\
\textsc{Main} \mathrel{\widehat=}\ & \mu N \bullet Get\_dis;\ N
\end{aligned}$$

Personal agent

The personal agent maintains the user's profile, including user's name, office location, interests, etc. (modeled as [NAME, SUBJECT, …]). After receiving a potentially interesting talk notification from the talk discovery agent (defined later), the personal agent uses operation *Check* to communicate with his calendar agent to check whether the user is free or not and with the MapQuest agent to ensure the talk will be held nearby. In our system we assume that a user only wants to attend the talks located within five miles from his office. If the user could attend the talk, the personal agent will inform the discovery agent and connect the calendar agent to update the user's timetable.

$$\begin{array}{l}
\textit{Person} \\
\hline
\quad \begin{array}{|l}
id : PID \\
name : NAME \\
office : PLACE \\
interests : \mathbb{P}\, SUBJECT \\
upd, talkch, dist, checktm : \mathbf{chan}
\end{array} \\
\\
\quad Check \mathrel{\widehat=} [tk : Talk, tresult : Status] \bullet talkch?(id, tk) \rightarrow \\
\qquad\qquad ((checktm!(id, tk.dt) \rightarrow\bullet\ checktm?tresult \rightarrow \textsc{Skip}) \parallel \\
\qquad\qquad\ (dist!(office, tk.place) \rightarrow [dresult : \mathbb{R}^{+}] \bullet \\
\qquad\qquad\qquad dist?dresult \rightarrow \textsc{Skip})); \\
\qquad\qquad [tresult = FREE \land dresult < 5] \bullet talkch!(id, GO) \\
\qquad\qquad\qquad \rightarrow upd!(id, tk.dt, BUSY) \rightarrow \textsc{Skip} \\
\qquad\qquad \Box\ [tresult = BUSY \lor dresult \geqslant 5] \bullet\ talkch!(id, NO) \\
\qquad\qquad\qquad \rightarrow \textsc{Skip} \\
\quad \textsc{Main} \mathrel{\widehat=} \mu\, N \bullet Check;\ N \\
\hline
\end{array}$$

Talk discovery agent

Schema *Talk* is defined for a general talk type. The *interested_subjects* records the interested subjects for the users. The talk discovery system senses market updates, finding new talks information. Once a new talk is found, it sends a notification to all the users who may be interested.

4. Extracting OWL Web ontology from the TCOZ model

It is important to have a thoroughly designed ontology since it will be shared by different agents and it forms the foundation of all agents' services. However designing a clear and consistent ontology is not a trivial job. It is useful to have some tool support in designing the ontology.

$$\begin{array}{|l}
\textit{Talk} \\
\hline
place : PLACE \\
dt : DateTime \\
subject : \mathbb{P}\, SUBJECT \\
\hline
\end{array}$$

$$notify\ ::=\ GO \mid NO \qquad\qquad \begin{array}{|l} interested\_subjects : \\ \quad Person \leftrightarrow SUBJECT \end{array}$$

$$\begin{array}{l}
\textit{Discovery} \\
\hline
\quad \begin{array}{|l}
users : \mathbb{P}_1\, Person \\
talkch : \mathbf{chan} \\
monitor : Talk\ \mathbf{sensor} \\
\hline
\forall\, u_1, u_2 : users \bullet u_1 \neq u_2 \Rightarrow u_1.id \neq u_2.id
\end{array} \\
\\
\quad \textsc{Main} \mathrel{\widehat=} \mu\, M \bullet [t : Talk] \bullet monitor?t \rightarrow ||| \ [u : users] \bullet \\
\qquad\quad ([interested\_subjects(\!|\ \{u\}\ |\!) \cap t.subject \neq \varnothing] \bullet \\
\qquad\qquad\quad talkch!(u.id, t) \rightarrow \\
\qquad\qquad\qquad [response : notify] \bullet talkch?(u.id, response) \rightarrow \textsc{Skip} \\
\qquad\quad \Box\ [interested\_subjects(\!|\ \{u\}\ |\!) \cap t.subject = \varnothing] \bullet \textsc{Skip});\ M \\
\hline
\end{array}$$

In this section, we demonstrate the development of an XSL application that automatically extracts the ontology related domain properties from the static aspects of TCOZ formal models (encoded in ZML format (Sun, 2001b)). The ontology for the system can be resolved readily from the static parts of TCOZ design documents. In the next section, we will demonstrate tools to automatically extract the semantic markup for services from dynamic aspects of TCOZ formal models.

ZML is an XML environment for Z family notations (Z/Object-Z/TCOZ). It encodes the Z family of documents in XML, so that the formal model can be easily browsed by the Web browser (e.g. Internet Explorer). The eXtensible Stylesheet Language (XSL) is a stylesheet language to describe rules for matching and translating XML documents. In our case we translate the ZML to OWL and OWL-S. The main process and techniques for the translation are depicted by Figure 1.

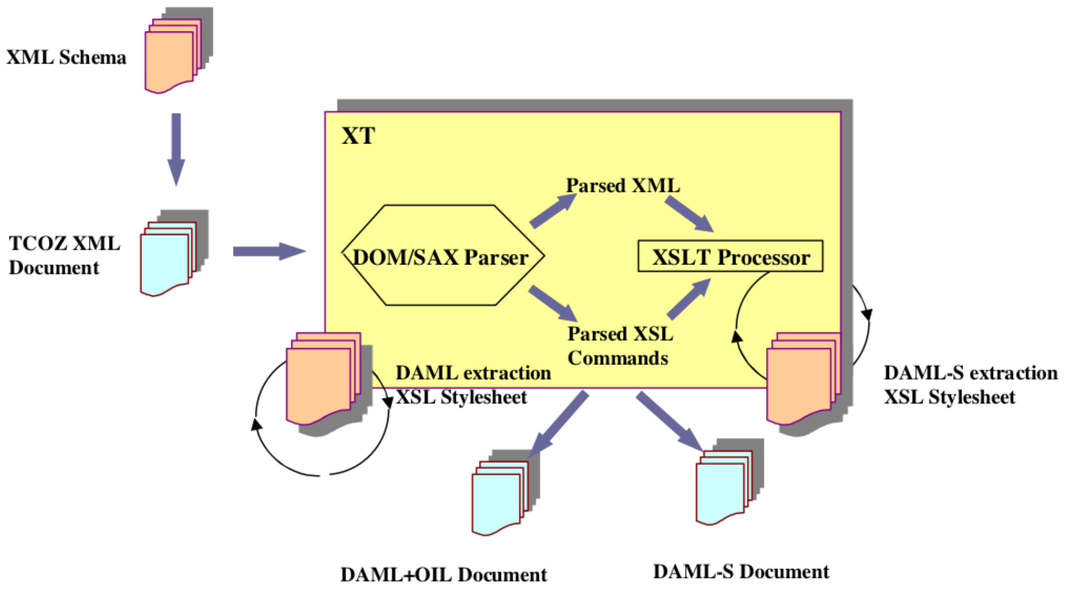


Figure 1: TCOZ OWL/OWL-S projection

A set of translation rules translating from TCOZ model (in ZML) to OWL ontology is developed in the following presentation.

Given type translation

The given types in the TCOZ model are directly translated into OWL/RDFS classes. This rule is applicable to the given types defined in both inside and outside of a class definition. The translation can be expressed as the following rule:

$$\frac{[T]}{T \in OWL\_class}$$

For example, the given type *PID* can be translated into a class in OWL with *PID* as ID.

```
Class(PID partial)
```

Axiomatic (Function and Relation) definition translation

$$\frac{\begin{array}{|l} R : B \leftrightarrow (\rightarrow) C \\ \hline \ldots \end{array} \qquad B, C \in OWL\_class}{R \in OWL\_objectproperty[B \rightarrow C]}$$

The translation from functions and relations in TCOZ to OWL ontology requires several cases. The relation *R* will be translated into an OWL property with *B* as the domain class and *C* as the range class. For total functions we translate it into an *owl:FunctionalProperty*.
In our talk discovery example, the relation *interested_subjects* can be translated into OWL as:

```
ObjectProperty(interested_subjects
     domain(person) range(subject))
```

Z Axiomatic (Subset and Constant) definition translation

Subset

In this situation, if *N* corresponds to an OWL class, then *M* will be translated into an OWL subclass of *N*. If *N* corresponds to an OWL property, then *M* will be translated into an OWL subproperty of *N*. The translation rules for the subset are shown as:

$$\frac{\begin{array}{|l} M : \mathbb{P}\, N \\ \hline \ldots \end{array} \qquad N \in OWL\_class}{M \in OWL\_subclass[N]}$$

$$\frac{\begin{array}{|l} M : \mathbb{P}\, N \\ \hline \ldots \end{array} \qquad N \in OWL\_objectpropert}{M \in OWL\_subproperty[N]}$$

Constant

In this situation, *X* will be translated into an instance of *Y*. The following is the translation rule:

$$\frac{\begin{array}{|l} x : Y \\ \hline \ldots \end{array} \qquad Y \in OWL\_class}{x \in instantceof[Y]}$$

Z state schema translation

A Z state schema can be translated into an OWL class. Its attributes are translated into OWL properties with the schema name as domain OWL class and the Z type declaration as range OWL class. In order to resolve the name conflict between same attribute names used in different schemas, we use the schema name appended with attribute name as the ID for the OWL property:

$$\begin{array}{|l} \hline S \\ x : T_1;\ \ y : \mathbb{P}\, T_2 \\ \hline \ldots \\ \hline \end{array} \qquad T_1, T_2 \in OWL\_clas$$

$$\frac{}{\begin{array}{l} S \in OWL\_class,\ S\_x \in OWL\_objectproperty[S \rightarrow T_1], \\ S\_y \in OWL\_objectproperty[S \leftrightarrow T_2] \end{array}}$$

For example the Talk schema defined in a previous section can be translated to OWL as:

```
Class(talk partial)
Class(place partial)
ObjectProperty(talk_place Functional
      domain(talk) range(place))
… …
ObjectProperty(talk_subject
      domain(talk) range(subject))
```

Class translation

An Object-Z class can be translated into an OWL class. Its attributes defined in state schema are translated into OWL properties with the class name as domain OWL class and the type declaration as range OWL class. Other translation details are similar to the Z state schema translation defined above.

$$\begin{array}{|l} \hline C \\ \begin{array}{|l} \hline x : T_1;\ \ y : \mathbb{P}\, T_2 \\ \hline \ldots \\ \hline \end{array} \\ \ldots \\ \hline \end{array} \qquad T_1, T_2 \in OWL\_clas$$

$$\frac{}{\begin{array}{l} C \in OWL\_class,\ C\_x \in OWL\_objectproperty[C \rightarrow T_1] \\ C\_y \in OWL\_objectproperty[C \leftrightarrow T_2] \end{array}}$$

For example the *Person* class defined in a previous section can be translated to OWL as:

```
Class(person partial)
ObjectProperty(person_id Functional
      domain(person) range(PID))
```

The predicates defined in Object-Z class invariant can be translated into OWL class restriction. For example, suppose that we add the predicate '# interests ≤10' in the class *Person* to denote that a person can at most register 10 interested subjects to the system. This will be translated to:

```
Class(person partial
    restriction(interests maxCardinality(10)
```

Note that as OWL is less expressive than TCOZ, not all the predicates defined in TCOZ can be translated. Other translation rules are omitted, as the aim of this chapter is to demonstrate the approach rather than providing the complete XSL program design.

The translation between TCOZ and OWL is not trivial. Rigorous study has been made to avoid the conflict between those two semantics. For example, the *schema inclusion* and *class inheritance* do not correspond to OWL subclasses relationship even though they appear to be the case. The reason is that, based on the semantics of Z schema and Object-Z class, an Object-Z class and its subclass has the disjoint instances set:

$$\forall c_1, c_2 : \mathit{Class} \bullet c_2 \ \underline{\mathit{inherit}}\ c_1 \Rightarrow c_2 \cap c_1 = \varnothing$$

This is totally different from the OWL subclass relationship, where every instance of an OWL class is the instance of its super class also. In the early work (Dong, 2004), it shows a Z semantics of DL and (Zucanu, 2006) shows that the consistency between the Z Semantics for the Semantic Web languages and the original OWL semantics.

5. Extracting OWL-S ontology from the TCOZ model

In the previous two sections we demonstrated how TCOZ could be used to capture the requirements of Semantic Web applications and how to project TCOZ models to OWL ontologies automatically. OWL ontology is used to define the common understanding for certain concepts. The dynamic aspects of Semantic Web services, which define what the service does and how it behaves is also crucial. Recently, OWL-S emerges to define such information for SW services. Extracting the semantic markup information (i.e. OWL-S) for a Semantic Web service from the formal requirements model is another important research work. In this section, we will demonstrate the development of another XSL program to automatically extract OWL-S information from TCOZ formal models. The semantic markup for the system can be resolved from the TCOZ design documents also.

Translation rules

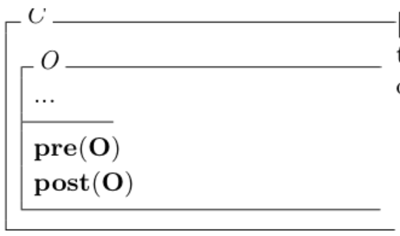
A set of translation rules translating from TCOZ model to OWL-S semantic markup for Semantic Web services are developed in the following:

Basic rule 1 (R1):

Each operation in TCOZ is modeled as a process (AtomicProcess or CompositeProcess) in OWL-S. In TCOZ, operations are discrete processes which specify the computation behaviors and interaction behaviors. From a dynamic view, the state of an object is subject to change from time to time according to its interaction behaviors, which are defined by operation definitions. At the same time the service process allows one to effect some action or change in the world. The connection between operations in TCOZ and service process in Semantic Web services is obvious. In order to resolve the name conflict between the same operation names used in different classes we use the class name appended with operation name as the ID for the process.

Basic rule 2 (R2):

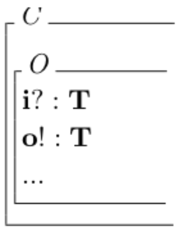
In the case that an operation invokes no other operations, the operation is translated as an AtomicProcess. A precondition appearing in a TCOZ operation schema definition is modeled as *precondition* in the respective service process. A postcondition appearing in a TCOZ operation schema definition is modeled as *effect* in the respective service process. The transformation is shown as below.



[pre(O) is a precondition of the operation O. post(O) is a postcondition of the operation O.]

$C\_O \in OWL - S\_AtomicProcess, \ C\_O\_pre(O) \in OWL - S\_precondition$
$C\_O\_post(O) \in \text{OWL-S\_effect}$

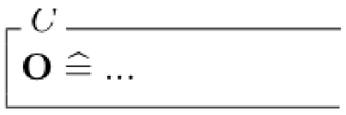
Basic rule 3 (R3):



$C\_O\_i \in \text{OWL-S\_input}, \ C\_O\_o \in \text{OWL-S\_output}$

An input appearing in a TCOZ operation schema definition is modeled as *input* in the respective service process. An output appearing in a TCOZ operation schema definition is modeled as *output* in the respective service process.

Basic rule 4 (R4):



$C\_O \in \text{OWL-S\_CompositeProcess}$

In the case that an operation calls other operations, the operation is translated as a composite process.

Basic rule 5 (R5):

Communication in TCOZ is modelled as an atomic process with input or output. In OWL-S, atomic processes, in addition to specifying the basic actions from which larger processes are composed, can also be thought of as the communication primitives of a (abstract) process specification.

$$\frac{\begin{array}{|l}\hline C \\ O \mathrel{\widehat{=}} \ldots \mathbf{Ch?i} \to \ldots \\ \hline\end{array}}{C\_O\_Ch \in \text{OWL-S\_AtomicProcess} \wedge C\_O\_Ch\_i \in \text{OWL-S\_input}}$$

$$\frac{\begin{array}{|l}\hline C \\ O \mathrel{\widehat{=}} \ldots \mathbf{Ch!o} \to \ldots \\ \hline\end{array}}{C\_O\_Ch \in \text{OWL-S\_AtomicProcess},\ C\_O\_Ch\_o \in \text{OWL-S\_output}}$$

Basic rule 6 (R6):

Each TCOZ process primitive will be translated into the proper OWL-S composite process. For example, the following two rules show how the translation is done for the sequential and parallel processes in TCOZ. Other translation rules for process primitive are omitted due to the limited space.

$$\frac{\begin{array}{|l}\hline C \\ O \mathrel{\widehat{=}} P_1;\ P_2 \\ \hline\end{array} \quad [P_1 \text{ and } P_2 \text{ are process components}]}{C\_O \in \text{OWL-S\_Sequence}[P_1, P_2]}$$

$$\frac{\begin{array}{|l}\hline C \\ O \mathrel{\widehat{=}} P_1 \parallel P_2 \\ \hline\end{array} \quad [P_1 \text{ and } P_2 \text{ are process components}]}{C\_O \in \text{OWL-S\_Split}[P_1, P_2]}$$

Basic rule 7 (R7):

$$\frac{\begin{array}{|l}\hline C \\ O \mathrel{\widehat{=}} [G].. \\ \hline\end{array}}{C\_O\_G \in \text{OWL-S\_precondition}}$$

The guards in TCOZ model are used to control the input of an operation. The guards are modelled as preconditions. Other translation rules are omitted, as the aim of this chapter is to demonstrate the approach rather than providing the complete XSL program design.

Case study

The *PIDManager* class defined for the Calendar agent will be used to demonstrate the translation. The *PIDManager* class has five operations, *AddPID*, *RemovePID*, *New*, *Delete* and *Validate*. Each of them will be translated into an OWL-S *process*.

The operation *AddPID* is an operation invokes no other operations, so it will be translated as an AtomicProcess (R2) with some standard header information.
The following shows part of the semantic markup for service AddID in OWL-S surface syntax[2]. The OWL-S code in RDF format is omitted here.

```
define atomic process PIDManager_AddPID(
  inputs: (PIEManager_AddPID_id - PID)
  result: PIDManager_addPID_Effect ))
```

AddPID has one input *id?* declared to be of type *PID*. It will be translated into *input* (*PIDManager_AddPID_id*) in OWL-S (R3). The operation *AddPID* has one predicate '*ids'=ids* $\cup$ *{id?}' (*`PIDManager_addPID_Effect`*)* which is a postcondition. It will be translated into *effect* (*PIDManager_AddPID_EFFECT*) in OWL-S (R2). The operation *RemovePID* can be translated similarly.

The operation *New* calls the other operation *AddPID*, so it is translated as a composite process (R4). It performs two subprocesses *PIDManager_AddPID_add_id_in* and *PIDManager_AddPID* in sequence. The *PIDManager_AddPID_add_id_in* process represents the communication on channel *add* (R5). The guard of the operation is translated as the precondition (*IDnotInIDS*)(R7).

The following shows the part of the semantic markup OWL-S for the operation *New*. The operation *Delete* and *Valide* can be similarly translated.

```
define atomic process add_id_in(
  inputs: (add_id - PID)
  precondition: IDnotInIDS …))
define composite process PIDManager_New(
  input:… …; result:……)
  {perform add_id_in (… …)
  ; // sequence
  PIDManager_AddPID (… …)
  }
```

6. Conclusion

In this chapter, we have demonstrated that TCOZ can be used as a high-level design language for modeling semantic web service ontologies and functionalities. Another major contribution of this chapter is that it develops systematic transformation rules and tools, which can automatically map TCOZ models to an OWL ontologies and OWL-S service description.

Other work (Dong, 2004) has recently investigated the development of Z semantics for the ontology language, OWL-DL, and automatic transformation of OWL and RDF ontologies into Z specifications. This allows us to use Z tools, such as Z/EVES, to provide a checking and verification environment for Web ontologies. This contrasts with the work presented in

[2] http://www.daml.org/services/owl-s/1.2/owl-s-gram/owl-s-gram-htm.html

this chapter, where we have investigated how RDF and OWL can be used to build a Semantic Web environment for supporting, extending and integrating various formal specification languages (Dong, 2002a). One additional benefit is that RDF query techniques can facilitate formal specification comprehension.

To summarise, we have demonstrated a clear synergy between Semantic Web and Formal Methods, showing how each can greatly support each other. This has been demonstrated through the ITTalks use case, and exploits the synergy between the two.

7. Future Research Directions

The rules and tool presented in this chapter allows us to build domain ontologies and service markup more easily Apart from forming the initial model used an OWL ontology and the OWL-S service, the TCOZ formal design model can provide an unambiguous requirement for the semantic web service system and the series of related supporting tools (Dong, 2002b; Sun, 2001a; Sun 2001b; Sun, 2003) can ensure the high quality of the design model. Furthermore, the formal model may also be used in such a way that can lead towards a suitable Web service implementation. The automatic generation of code from formal specifications is a popular research area, which has received considerable attention, and in which many of tools and systems already exists (Woodcock, 1996). However the refinement from formal models to Web Service specific implementations is a relatively new research area. The details of the refinement calculus are beyond the scope of this chapter and will be address in a separate paper.

References

Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., Martin, D., McIlraith, S., Narayanan, S., Paolucci, M., Payne, T., & Sycara, K. (2002). DAML-S: Web Service Description for the Semantic Web. *In First International Semantic Web Conference (ISWC) Proceedings*, pages 348–363.

Bussler, C. (2001). B2B protocol standards and their role in semantic b2b integration engines. *Bulletin of the Technical Committee on Data Engineering, 24(1*), pages 67 - 72.

Berners-Lee, T., Hendler J., & Lassila O. (2001). *The semantic web*. ScientificAmerican.

Brickley, D., & Guha, R.V. (editors). (2004). RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/.

Cost, R., Finin, T., & Joshi, A. (2002). Ittalks: A case study in the semantic web and DAML. In *proceedings of the International Semantic Web Working Symposium*, pages 40 – 47.

Dean, M., Connolly, D., Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., & L. A. S. (editors). (2004). OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref/.

Dong, J. S., Lee, C. H., Li, Y. F., & Wang, H. (2004) Verifying daml+oil and beyond in z/eves. In *The 26th International Conference on Software Engineering (ICSE'04)*, pages 201 – 210. IEEE Press.

Dong, J. S., & Mahony, B. (1998) Active Objects in TCOZ. In *The 2nd IEEE International Conference on Formal Engineering Methods (ICFEM'98)*, pages 16–25. IEEE Press.

Dong, J. S., Sun, J., & Wang, H. (2002a). Semantic Web for Extending and Linking Formalisms. In L.-H. Eriksson and P. A. Lindsay, editors, *Proceedings of Formal Methods Europe: FME'02*, pages 587 – 606, Copenhagen, Denmark, Springer-Verlag.

Dong, J. S., Sun, J., & Wang, H. (2002b) Z Approach to Semantic Web Services. In *International Conference on Formal Engineering Methods (ICFEM'02),* Shanghai, China. LNCS, Springer-Verlag.

Dong, J. S., Li, Y. F., Sun, J., Sun, J., & Wang, H. (2002c) Xml-based static type checking and dynamic visualization for TCOZ. In *4th International Conference on Formal Engineering Methods,* pages 311–322. Springer-Verlag.

Duke, R. & Rose, G. (2000). *Formal Object Oriented Specification Using Object-Z*. Cornerstones of Computing. Macmillan.

Heijst, G. V., Schreiber, A. T., & Wielinga, B. J. (1997) Using explicit ontologies in KBS development, *Int. J. Hum.-Comput. Stud.*, vol. 46, pp. 183-292.

Lieberman, H., Nardi, B. A. & Wright, D. J. (2001) Training agents to recognize text by example. *Autonomous Agents and Multi-Agent Systems,* 4(1-2):79–92.

Mahony, B. & Dong, J. S. (1999) Sensors and Actuators in TCOZ. In J. Wing, J. Woodcock, and J. Davies, editors, *FM'99: World Congress on Formal Methods*, Lect. Notes in Comput. Sci., pages 1166–1185, Toulouse, France, Springer-Verlag.

Mahony, B. & Dong, J. S. (2000) Timed Communicating Object Z. *IEEE Transactions on Software Engineering*, 26(2):150–177.

Roman, D., Keller, U., Lausen, H., Bruijn, J.D., Lara, R, Stollberg, M., Polleres, A., Feier, C., Bussler, C., & Fensel, D. (2005) Web services modeling ontology. *Journal of Applied Ontology*, 39(1):77–106.

Schneider, S. & Davies, J. (1995) A brief history of Timed CSP. *Theoretical Computer Science*, 138: 243 – 271.

Smith, G. (2000) The Object-Z Specification Language. *Advances in Formal Methods*. Kluwer Academic Publishers.

Studer, R., Benjamins, V., & Fensel, D. (1998) Knowledge engineering, principles and methods. Data and Knowledge Engineering, 25(1-2):161–197.

Sun, J., Dong, J. S., Liu, J., & Wang, H. (2001a) A XML/XSL Approach to Visualize and Animate TCOZ. In J. He, Y. Li, and G. Lowe, editors, *The 8th Asia-Pacific Software Engineering Conference (APSEC'01),* pages 453–460. IEEE Press.

Sun, J., Dong, J. S., Liu, J., & Wang, H. (2001b) Object-Z Web Environment and Projections to UML. In *WWW-10: 10th International World Wide Web Conference*, pages 725–734. ACM Press.

Sun, J. (2003) Tools and Verification Techniques for Integrated Formal Methods. PhD thesis, National University of Singapore.

Woodcock, J. & Davies, J. (1996) *Using Z: Specification, Refinement, and Proof,* Prentice-Hall International.

Zave, P. & Jackson, M. (1997) Four dark corners of requirements engineering. *ACM Trans. Software Engineering and Methodology*, 6(1):1–30.

Zucanu, D., Li, Y. F. & Dong, J. S. (2006) Semantic Web Languages - Towards an Institutional Perspective. In *Algebra, Meaning, and Computation: A Festschrift in Honor of Prof. Joseph Goguen*, Eds. Futatsugi, Jouannaud, and Meseguer, pages 99-123, LNCS 4060, Springer-Verlag.

Additional Reading

Readers who are interested in more detailed information about formal notations used in this chapter will find that there is a wide selection of resources available. The following publications are good, basic texts:

- Duke, R. & Rose, G., Formal Object Oriented Specification Using Object-Z . Cornerstones of Computing Series (editors: R. Bird, C.A.R. Hoare), Macmillan Press, March 2000.
- Woodcock, J. & Davies, J., Using Z: Specification, Refinement, and Proof. Prentice-Hall, 1996
- Davies, J., Specification and Proof in Real-Time CSP, Cambridge University Press, 1993.
- Hoare, C. A. R. Communicating Sequential Processes. Prentice-Hall International, 1985.