# Transitioning Applications to Semantic Web Services: An Automated Formal Approach

Hai H. Wang Nick Gibbins Terry Payne Ahmed Saleh
School of Electronics and Computer Science
University of Southampton, Southampton, UK
{hw, nmg, trp, amms}@ecs.soton.ac.uk
Yuang-Fang Li
School of ITEE, The University of Queensland
Queensland, AU
liyf@itee.uq.edu.au

**Abstract:** Semantic Web Services [MSZ01] have been recognized as a promising technology that exhibits huge commercial potential, and attract significant attention from both industry and the research community. Despite expectations being high, the industrial take-up of Semantic Web Service technologies has been slower than expected. One of the main reasons is that many systems have been developed without considering the potential of the web in integrating services and sharing resources. Without a systematic methodology and proper tool support, the migration from legacy systems to Semantic Web Service-based systems can be a very tedious and expensive process, which carries a definite risk of failure. There is an urgent need to provide strategies which allow the migration of legacy systems to Semantic Web Services platforms, and also tools to support such a strategy. In this paper we propose a methodology for transitioning these applications to Semantic Web Services by taking the advantage of rigorous mathematical methods. Our methodology allows users to migrate their applications to Semantic Web Services platform automatically or semi-automatically.

## Introduction

The recent uptake of automated services over the Internet and World Wide Web has pushed the boundaries of Distributed Systems, by facilitating the greater proliferation of disparate, sharable resources such as computer systems and software applications, and the pragmatic uptake of interconnectable services, provided by a variety of different service providers. Software applications have evolved from monolithic, stove-pipe applications to loosely federated, interacting services that are dependent on networked resources to provide optimal functionality. This is largely due to a change in the perception of current software engineering practices, from using local functions and objects as software building blocks, to distributed, encapsulated, independent components. The emergence of *Web Services*, i.e. web-accessible programs that now proliferate the World Wide Web by providing user access to applications supporting tasks such as e-commerce, entertainment, etc, have greatly facilitated this migration for both enterprise and Grid-based applications due to the near ubiquitous World-Wide-Web infrastructure, cross-platform interoperability, and de-facto Web standards for syntax, addressing, and communication protocols.

The Semantic Web [BLHL] is becoming increasingly popular because it proposes an evolution of the current Web from a web of documents to a distributed and decentralized, global knowledge-base. The realization of the Semantic Web has facilitated the markup and manipulation of complex taxonomic and logic relations

between entities published on the Web. A fundamental component of the Semantic Web will be the formal markup and subsequent discovery and machine-comprehension of Web services. By semantically annotating the relevant aspects of declarative Web Service descriptions in a machine-readable format that can facilitate logical reasoning, such service descriptions become interpretable based on their meanings, rather than simply on a symbolic representation. The advantage of this is that many of the tasks involved in using Web Services can be (semi-) automated, for example: discovery, selection, composition, mediation, execution, monitoring, etc. Thus, Semantic Web Service Research [MSZ01] has been recognized as one of the most promising technologies to emerge, exhibiting huge commercial potential, and attracting significant attention from both industry and the research community.

Despite its great prospect of success, the industrial take-up of Semantic Web Services technologies has been slower than expected. This was mainly due to the fact that many systems have been developed without considering the potential of the Web for integrating services and sharing resources. Without a systematic methodology and proper tool support, the migration from legacy systems to Semantic Web-Service based systems could be a very tedious and expensive process, which carries a definite risk of failure. There is an urgent need to provide strategies which allow the migration of legacy systems to Semantic Web Services platforms and also tools to support such a strategy.

In this paper we propose a methodology for automatcially/semi-automatically transitioning legacy applications to Semantic Web Services by adopting a formal approach. Such formal methods include mathematically rigorous techniques and tools for the specification, design and verification of software and hardware systems. A formal language has a well-defined syntax and semantics, which facilitate the use of automated processing. In addition, formal methods have associate calculation rules that can be used to analyze specifications in order to determine correctness and consistency.

Our approach first utilizes reverse-engineering technologies to abstract a formal specification of a legacy system from its code implementation. This formal specification gives the user a good understanding and a simple description of the system. The correctness of this specification can be verified by using various formal validation and verification tools. Then, we develop a set of rules and a tool to automatically generate domain ontologies and service descriptions used by Semantic Web Service systems from the formal specification. Finally, formal refinement techniques are applied to generate the new equivalent Web service implementation. Our approach ensures that the functionalities of existing systems are correctly migrated and the transitioning process is carried out automatically or semi-automatically.

The remainder of this paper is organized as follows. Section 2 briefly introduces the background material in the areas of formal methods and Semantic Web Services. Section 3 summarizes the major challenges for migrating a legacy system to Semantic Web Service platform. Section 4 concentrates on the different phases of our methodology. Section 5 evaluates our aproach. Section 6 concludes the paper and discusses possible future work.

# Overview

## Semantic Web & Semantic Web Services

The Semantic Web is an extension of the current World Wide Web, which embeds *knowledge* in the form of semantic annotations within web pages. The inclusion of content with a well-defined meaning has meant that documents and resources published on the web can be more easily accessible by computer programs, thus better enabling computers and people to work in cooperation. HTML, the current Web data standard, is aimed at delivering information to the end user for human-consumption (e.g. display this document). XML is aimed at delivering data to systems that can understand and interpret the information. XML is focused on the syntax (defined by the XML schema or DTD) of a document and it provides essentially a mechanism to declare and use simple data structures. However there is no way for a program to actually understand the knowledge contained in the XML documents.

The Resource Description Framework (RDF) [LE99] is a foundation for processing metadata; it provides interoperability between applications that exchange machine-understandable information on the Web. RDF uses XML to exchange descriptions of Web resources and emphasizes facilities to enable automated processing. The RDF descriptions provide a simple ontology system to support the exchange of knowledge and semantic information on the Web. RDF Schema [D. 04] provides the basic vocabulary to describe RDF vocabularies, and can be used to define properties and types of the web resources. In this respect, RDF Schema plays a similar role to XML Schema; XML Schema gives specific constraints on the structure of an XML document, while RDF Schema provide information about the interpretation of the RDF statements.

The Semantic Web, by its very nature, is highly distributed, and thus different parties may have different understandings of the same concept. Ideally, a program must have a way of discovering common meanings from different understandings. These common meanings are a key concept in Semantic Web systems ,and are known as *Ontologies*. Ontologies are an explicit, formal specification of a shared conceptualisation of a domain, and provide a machine-readable, and agreed-upon representation of the conceptual vocabulary used to represent a domain of discourse in applications. Though ontologies can be very expressive, the most typical kind of ontology found on the Web is normally epistemic or taxonomic, and typically includes a simple set of inference rules. The use of ontologies can enhance the functioning of the Web in many ways.

OWL [BvHH+04] is a recently standardized ontology language, developed by members of the World Wide Web Consortium[1] and the Description Logic community. An OWL ontology consists of classes, properties and individuals. Classes are interpreted as sets of objects that represent the individuals in the domain of discourse. Properties are binary relations that link individuals, and are interpreted as sets of tuples, which are subsets of the cross product of the objects in the domain of discourse.

[1] http://www.w3.org

OWL classes fall into two main categories -- named classes and anonymous (unnamed) classes. Anonymous (unnamed) classes are formed from logical descriptions. They contain the individuals that satisfy the logical description. Anonymous classes may be sub-divided into restrictions and 'logical class expressions'. Restrictions act along properties, describing sets of individuals in terms of the types of relationships that the individuals participate in. Logical classes are constructed from other classes using the boolean operators *AND*, *OR* and *NOT*.

A fundamental aim of the Semantic Web will be the markup of Web services to make them computer-interpretable, use-apparent, and agent-ready. 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, invocation, and composition and interoperation, and to that end allows the definition of three essential types of knowledge about a service: the profile, the process model and the grounding. The *profile* describes what the service does, the *process model* describes how the service works, and the *grounding* describes how the service is to be used. The OWL-S process model is intended to provide a basis for specifying the behavior of a wide array of services, and enables planning, composition and agent/service inter-operation. There are two key components of an OWL-S process model: the process, and the process control model. The process describes a Web Service in terms of its input, output, precondition, effects, and where appropriate, its component subprocess. 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*.

## Z

Z notation is a state-oriented formal specification language based on set theory and predicate logic. A Z specification typically includes a number of state and operation schema definitions. A state schema encapsulates variable declarations and related predicates (invariants). The system state is determined by values taken by variables subject to restrictions imposed by state invariants. An operation schema defines the relationship between the 'before' and 'after' states corresponding to one or more state schemas. Complex schema definitions can be composed from the simple ones by schema calculus. Z has been widely adopted to specify a range of software systems (see ). Various tools, i.e. editors, type/proof checkers and animators, have been developed for Z.

Consider the Z model of a stack. Let the given type *Item* represent a set of items. The notation for this is:

$[Item]$ [item type]

The stack contains operations to pop items off and push items onto the stack. The total items in the stack cannot be more than *max* (say, a number larger than 100). The global constant *max* can be defined using the Z axiomatic definition as:

$$max : \mathbb{N}$$
$$max > 100$$

The state, potential state change and initial state of the stack system can be specified in Z as:

*Stack*
$$items : \text{seq } Item$$
$$\#items \leq max$$

*StackInit*
$$Stack$$
$$items = \langle\rangle$$

The operations to push items on, and pop items off of the stack can be modelled as:

*Push*
$$\Delta Stack$$
$$item? : Item$$
$$items' = \langle item? \rangle \frown items$$
$$\#items < max$$

*Pop*
$$\Delta Stack$$
$$item! : Item$$
$$items \neq \langle\rangle$$
$$items = \langle item! \rangle \frown items'$$

The contents of the upper half of a schema define the types of the variables used, and may include definitions from other schemas (e.g. the use of *Stack* in the definition of *StackInit*). The lower half of a schema defines the invariants that hold over the variables in the schema. The variable names in a operation schema are conventionally annotated with suffixes to indicate whether they refer to the state of the variable after the execution of the operation (e.g. *items'*), to an input variable (e.g. *item?*) or an output variable (e.g. *item!*).
More complex operations can be constructed by using schema calculus, e.g., a new item which is pushed on and then popped off, say *Transit*, can be specified by using the sequential composition schema operator ';' as:

$$Transit \mathrel{\widehat=} Push \fatsemi Pop$$

which is an (atomic) operation with the effect of a *Push* followed by a *Pop*. Other forms of schema calculus include schema conjunction '$\wedge$', disjunction '$\vee$', implication'$\Rightarrow$', negation '$\neg$'and pipe '>>', which have been discussed in many Z text books [Spi89, WD96].

## Major challenges

The objective of our methodology is to guide the migration of a legacy system to a Semantic Web Service system. Without a systematic methodology and proper tool support, the migration process could be very tedious and expensive, which carries a definite risk of failure. The challenges of this migration come from several aspects.

- It is difficult to correctly recognize all the functionalities of existing system. Many legacy systems do not have precise design documentations; and even there exists some documentations, quite often you would find that they were different from the actual codes. Since much of the functionality of existing software has been achieved over a period of time and implemented by various developers, it has to be preserved for many reasons. After being deployed for a few years, legacy systems are sometimes described by system maintainers as "they have performed some useful tasks, but I really do not understand why and how that happened".
- Most legacy systems are implemented using either procedural or object-oriented programming styles, which are different from the nature of the service-oriented paradigm adopted by Semantic Web Services. The service-oriented paradigm defines the use of *loosely coupled* software services to support the requirements of business processes and software users. Independent services have defined interfaces that can be called to perform their tasks in a standard way, without the service having prior knowledge of the calling application, and without the application being aware of how the service actually performs its tasks. By contrast, most legacy systems were built following *tightly coupled* point-to-point integration principles. Decomposing the existing tightly coupled systems at both functional and implementation levels is not an easy task.
- Designing a conceptualization (ontology) and markup of services for a particular domain is also not a trivial task. This is because:
    - The existing ontology and service markup languages are too low-level to be understood and used by domain experts. For example, we regard the underlying ontology languages, such as OWL, as an "assembly code" to be seen only by ontology experts. Domain experts should interact with "high level abstract languages".
    - The current practice in ontology and service markup development is at a similar stage to software development two decades ago. It assumes that each ontology and service markup starts from scratch, and it approaches the development more as a craft than as a principled engineering discipline. This has lulled the ontology community into a false confidence and led to knowledge engineers building ontologies on behalf of domain experts rather than enabling domain experts to develop their own ontologies for themselves. The ontology and service markup developers have to concentrate on both domain issues and low level modeling details.
    - Since the Semantic Web and Semantic Web Service research in general are still evolving, tool support for ontology and service markup design (though rudimentary) is also improving. The Semantic Web community is currently focussing on developing automatic tools to check the logical satisfiability of an ontology [HM01, Hor98]. However, logically satisfiability does not necessarily imply the correctness of the ontology, and the field lacks tools to assist users in the validation and verification of knowledge models.
- Implementing all the desired functionalities of systems in the Semantic Web Services platform requires much effort from experienced software engineers and programmers. When compared with many legacy software

environments, the Web is highly open and distributed. This brings to light new issues like software security, the interaction and integration of different software components, and so on. Furthermore, Web/Semantic Web application have lots of their own APIs, tools and protocols. Programmers have to learn these these new techniques.

## The approach

In this section we propose a methodology of transitioning legacy applications to Semantic Web Service systems by applying formal methods. Using formal methods, we aim, as far as possible, to automate mechanical tasks during the transitioning process. In terms of the transitioning process, a legacy system migration can be divided into several major phases. Each phase consists of a number of individual migration activities. Figure 1 shows the main steps of our methodology.

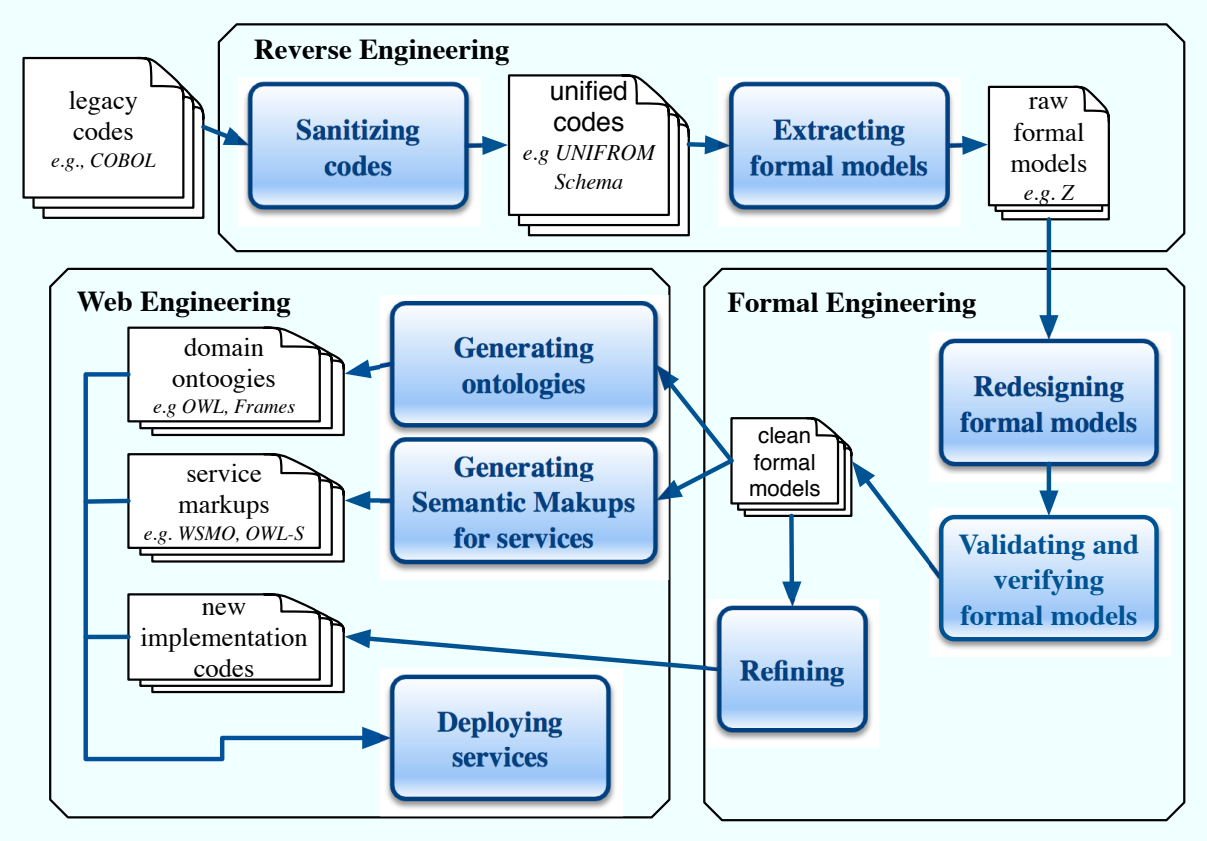


*Figure 1: The framework*

### Phase 1: Abstraction of formal specification from program code.

As mentioned in the previous section, one of the major challenges of transitioning legacy applications to the Semantic Web Services platform is that we often don't know what to do with existing software, even when we are sure that it performs a

useful job. In this phase, we use the reverse engineering techniques to extract a precise and abstract requirements model from the implementation code.
Reverse engineering (RE) is the process of discovering the technological principles of a system through analysis of its structure, function and operation [CC90]. Integrating formal methods in reversing engineering is a popular research area; there have been several investigations focusing on the use of rigorous mathematical methods for extracting formal specifications from existing code [GC93]. Some notable works include that by using category and monad theory, Lano developed a framework for abstracting high level specification[LB90]. [LW90] proposed an approach to identifying objects in procedural codes, where the characterization of candidate objects is based on recognizing common routines, operations, data types, and data items through the examination of global data and major data types. Haughton also investigated the identification of objects in procedural code as well as specification [HL91]. Figure 2 shows an example of extracting a fragment of Pascal code to a Z model. The code implements a function *mts* which checks if a stack is full. Details of this example, the respective formal model, and the extracting process can be found in [GC93].

```
St = record
     t:integer;
     e:array[1..maxlength] of elementtype
   end;
...
function mts(S:st):boolean;
   begin
      if S.t>maxlength then
         mts :=true
      else  mts := false
   end;
```

*(A) Pascal code*

```
...
proc mts: St →¬ boolean
in(s:St)
out(mts:boolean)
{pre:domain(S) }
{post:  (((S.t>maxlength) ∧
           (mts = true)) ∨
         (¬ (S.t > maxlength) ∧
           (mts = false))) ∧
         demain(S)}
```

*(B) Formal model*

*Figure 2: An example of extracting formal model from Pascal code.*

As mentioned before, there are a number of different approaches/tools for extracting formal models from code implementation. Users can choose the most suitable one for their needs. Depending on the source code of the legacy system, the target formal notation and the different support tool, the activities within this phase may vary. In this paper, we use the method proposed by the REDO project [BBL91] as an example to illustrate the possible activities that may take place during this phase.
REDO is a large collaborative ESPRIT project concentrating on reverse engineering, on the principle that applications are usually unmaintainable in the form in which they are presented for maintenance, and work has to be done in order to rediscover the required documentation and design information. The tool developed by REDO can assistant users to automatically or semi-automatically extract Z++ (a variety of Z) model from a variety of legacy code (e.g., COBOL, FORTRAN, C). This can be done in several steps. As shown in Figure 1, initially the legacy program (e.g., COBOL) is automatically analyzed and translated to UNIFORM code, and redundant control structures are eliminated. UNIFORM was developed by the REDO project as a kind of formal universal intermediate language, which abstracts features of programing languages, such as COBOL, FORTRAN and C. During this process, certain details of the implementation may be lost, such as whether

integers were stored as 16 or 32 bits, but all the essential functionality for operating and understanding the system is retained. The formal model (Z model) is then extracted from the UNIFORM code by first abstracting the UNIFORM code to a first order functional language, in which details of the algorithms used are lost in favor of implicit representation of functionality, and then transforming the functional language to a representation in Z with the users' assistance, during which more implementation details are lost [BBL93b].
By using a reverse enginerring approach, extracting formal specifications from legacy code can be beneficial to the Semantic Web Services transitioning process in several ways:

- Since the requirements model is derived directly from implementation code, it is able to show the latest information about all the system's functionalities.
- The system requirements model is specified by a formal notation, which allows users to understand the functionalities of the existing system more easily and precisely, without any ambiguity.
- The resulting requirements formal model focuses on the system's core functionalities at a high level, which means that many implementation details are ignored. Therefore, the model is more loosely coupled compared to the implementation code and can be more easily decomposed to services.

## Phase 2: Redesigning system, Verification and Validation of formal model

Migrating an application onto a Semantic Web Services platform requires revisiting some functionalities of the existing system and also adding new features, such as the management of security, communication with other web service agents etc. Even after the system has been successfully deployed as a Semantic Web Service, it may still need to evolve from time to time. We must ensure that the redesigned system is robust and correct. Furthermore, it is highly desirable to make it easier, safer and traceable to update and maintain the system with this evolving process. The use of formal technology has been shown to be effective in aiding software maintenance [BBL93a]. The developers should redesign the system in an iterative and incremental way. Depending on the usage, new functions can be identified. The new design piece is formalized and integrated into existing formal model; a Verification and Validation step is then always performed.
Verification and Validation (V&V) is the process of checking that a software system meets its specifications, and that it fulfills its intended purpose. Validation checks that the software satisfies or fits the intended usage — i.e., you built the right product, while verification is the act of proving or disproving the correctness of a system with respect to a certain mathematical specification or property. Being one of the most important research areas for formal methods communities during the last two decades, model verification and validation has led to the development of many mature formal reasoning tools, from type checkers and animators, to provers

etc[2]. In this phase of the transitioning process, users will use those tools to validate and verify the formal models resulted from the abstraction phase.
Designing and implementing the changes to systems by changing the formal specifications, and using formal tools to ensure the correctness of the formal model, can be beneficial to the Semantic Web Services transitioning process in a number of ways:

- The old system may contain errors. Formally reasoning and checking the model can expose them, and so avoid propagating the old flaws to the new system.
- It can ensure that all the added components can be cleanly integrated to the whole system without interfering with existing system features.
- Since the migration and evolving process have been formally documented, it can be easily maintained and traced.

## Phase 3: Generation of domain ontology and semantic Web service markup from specifications

The difficulty of designing and developing a high level domain ontology and semantic markup for services is a major obstacle for SW Services deployment. It requires the developers to have both domain knowledge and good skills in ontology engineering. In this phase, we will present a set of translation rules and a tool which can generate a domain ontology automatically and semantic service markup from the formal Z model.

### *Generation of domain ontology from specifications.*

| Z constructors | Z examples | OWL Constructors transformed to | OWL examples |
|---|---|---|---|
| Given Type | *[T]* | OWL Class | *Class(T);* |
| Axiomatic Relation | $R : B \leftrightarrow (\rightarrow) C$ … | Property | *Property(R domain(B) range(C));* |
| Subset | $M : \mathbb{P} N$ … | OWL Individual | *Class(M); if N is a class SubClass(M, N);* ***OR*** *ObjectProperty(M);if N is a property SubProperty(M, N);* |
| Constant | $x : Y$ … | OWL Individual | *Individual(x, type(Y));* |
| State Schema | $S$ $x : T_1;\ y : \mathbb{P}\, T_2$ … | OWL Class OWL Property | *Class(S); FunctionalProperty(S_x, domain(S) range($T_1$)); Property(S_y, domain(S), range($T_2$));* |

*Figure 3: Generation of OWL from Z Model*

[2] http://vl.zuser.org/#tools

The domain ontology used by an application can be generated from the static part of its specification. Figure 3 summaries some of the transformation rules.
Given types of the Z model are directly translated into OWL classes. Also, a relation defined in Z is translated into either an OWL property or OWL FunctionalProperty, depending on the functionality of the relation. Furthermore, the domain and range types of the relation are mapped to OWL domain and range axioms. The subset relation in Z is mapped to OWL subClass or subProperty axioms as appropriat, and Z constants are transformed to OWL individuals. 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 to the attribute name as the ID for the OWL property.
For example, the following Z model defines a schema *Trip* which has two attributes *origin* and *destinations*. A *trip* can only have one origin *place* and several destination places (*Place* is a Z given type). It also defines *tripInnVen* as one concrete *trip* from *Innsbruck* to *Southampton* and *Manchester*, where *Innsbruck*, *Southampton* and *Manchester* are constants with type *Place*.

$$
\begin{array}{|l}
\hline
\mathit{Trip} \\
\mathit{origin} : \mathit{Place} \\
\mathit{destinations} : \mathbb{P}\, \mathit{Place} \\
\ldots \\
\hline
\ldots \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\mathit{tripInnVen} : \mathit{Trip} \\
\hline
\mathit{tripInnVen}.\mathit{origin} = \\
\quad \mathit{Innsbruck} \\
\mathit{tripInnVen}.\mathit{destinations} = \\
\quad \{\mathit{Manchester}, \mathit{Southampton}\} \\
\hline
\end{array}
$$

An OWL ontology can be automatically generated from the above Z model. To save space, we choose to use DL syntax to represent the OWL ontology.

```
Class(    ...)
ObjectProperty(         domain(    )
     range(     ))
FunctionalProperty(              domain(    )
     range(     ))
Individual(tripInnVen type(    )
     value(         Innsbruck)
     value(              Manchester)
     value(              Southampton))
```

Due to the limited space, we only present a portion of the translation rules. The translation between Z and OWL is not trivial. Rigorous study has been made to avoid the conflict between the differing semantics of OWL and Z. For example, the *schema inclusion* and *class inheritance* do not correspond to OWL's subclass relationship, even though it initially appears to be so. The reason is that, based on Z semantics, a schema and its extended schema (via schema inclusion) are a disjoint data type set.

$$\forall s_1, s_2 : \mathit{Schema} \bullet s_2 \ \underline{\mathit{extended}}\ s_1 \Rightarrow s_2 \cap s_1 = \varnothing$$

This is totally different from the OWL schema extending relationship, where all the instances for an OWL subclass are also instances of its super class.
Currently, there has been much debate on the most suitable ontology definition languages for Web applications. Description logic-based ontology languages, such as OWL, are one major genre and frame-based ontology languages are another genre. Each of them has their advantages and disadvantages and can be found more applicable for certain use cases. In this paper, we only present a set of transformation rules from a Z model to an OWL ontology. However, the final version of the tool would allow users to import different sets of rules for their usages, such as translating a Z model to a frame ontology (adopted by WSMO).

## *Generation of Semantic Web service markup from specifications.*

In the previous step we showed that the domain ontology used by an application can be automatically generated from the static part of a Z specification. Now we demonstrate how the semantic markup of Web services can be extracted from the dynamic aspects of a Z specification. We will use OWL-S as an example to illustrate the translation process. The transformation to other Semantic Web service standards, such as WSMO and SAWSDL can be defined as well. Figure 4 presents some of the translation rules.

| **Z constructors** | **Z examples** | **OWL-S Constructors transformed to** | **OWL-S examples** |
|---|---|---|---|
| Simple Operation Schema | $OP$<br>$\Delta State$<br>...<br>... | Atomic Process | *define* ***atomic process OP*** *(...);* |
| Operation Inputs and Outputs | $OP$<br>$\Delta State$<br>...<br>... | Process inputs and outputs | *define atomic process OP* ***(inputs: (Op_i - T1),*** ***(outputs: (Op_o - T2),*** *... );* |
| Operation Preconditions and Postconditions | $OP$<br>...<br>**Pre(Op)**<br>**Pre(Op)** | Process preconditions and effects | *define atomic process OP* ***(precondition: Pro(Op),*** ***result: (Effect(op),*** *...) ... );* |
| Complex Operation Schema | $Op \mathrel{\widehat=} ...$ | Composite Process | *define* ***composite process OP*** *(...){...};* |
| Schema Composition | $Op_1 \fatsemi Op_2$ | Sequence process | *define composite process OP(...)* ***{perform Op1(...) ;*** ***perform Op2(...) }*** *;* |
| Schema Disjunction | $Op_1 \lor Op_2$ | Choice Process | *define composite process OP(...)* ***{perform Op1(...) ;?*** ***perform Op2(...) }*** *;* |
| Schema Conjunction | $Op_1 \land Op_2$ | Split Process | *define composite process OP(...)* ***{perform Op1(...) \|\|<*** ***perform Op2(...) }*** *;* |

*Figure 4: Generation of OWL-S from Z model*

Operations in Z specify both the computation and interaction behaviors. From a dynamic view, the state of an object is subject to change from time to time according to its interaction behavior, which is 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 Z and a service process in Semantic Web services is obvious. Each simple operation in Z is modeled as an atomic process in OWL-S. An input appearing in a Z operation schema definition is modelled as an *input* in the respective service process. Similarly, an output appearing in a Z operation schema definition is modelled as an *output* in the respective service process. A precondition appearing in a operation schema definition is modeled as a *precondition* in the respective service process, and a postcondition appearing in a operation schema definition is modelled as an *effect* in the respective service process. There exist algorithms and tools to calculate the preconditions and postconditions of an operation from the predicates. An operation defined by a Z schema operation is modeled as different type of composite process. For example, a Z schema composition is translated to an OWL-S sequence process.

The following Z model defines an operation for adding one destination place to a trip and the destination should not be too far away from the origin place. *NotTooFar* is defined to abstract the distance relationship between two places.

$$NotTooFar : Place \times Place$$

*AddCloseDestination*

$$\Delta Trip$$
$$newDestination? : Place$$

---

$$(newDestination?, origin) \in NotTooFar$$
$$destinations' = desination \cup \{newDestination?\}$$
$$origin' = origin$$

From this specification we can generate the following OWL-S ontology, where *Place*, *Trip*, *NotTooFar*, and *Trip_desinations* ect. are OWL classes and properties extracted from the Z model based on the rules defined in previous steps.

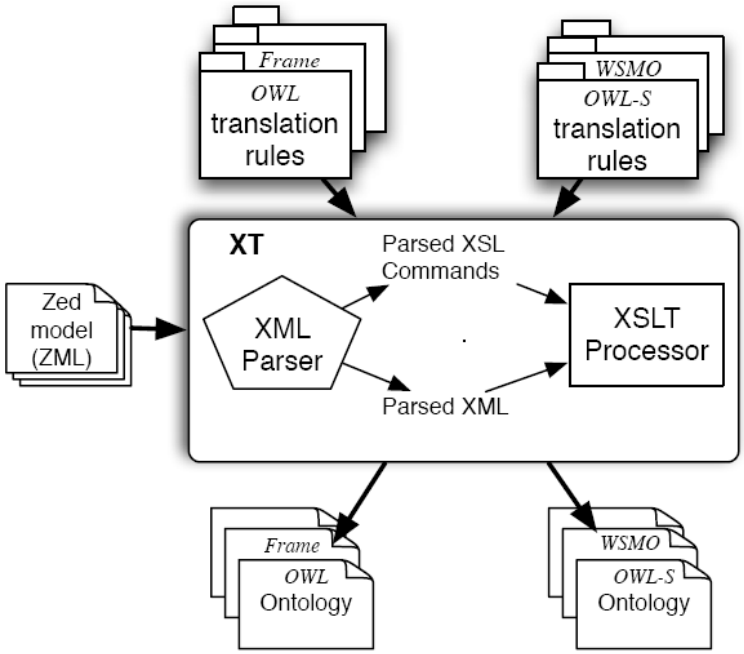


*Figure 5: Transformation tool design*

### *Automatic transformation tool*

In this subsection, we show the design of a tool which generates domain ontologies and Semantic Web Service markup from a formal Z model (encoded in ZML format [SDLW01]). This tool is realized by XSL technology. ZML is an international standard XML markup for Z specifications. It encodes the Z family documents in XML format so that the formal model can be easily browsed by any Web browser (e.g. Internet Explorer). The eXtensible Stylesheet Language (XSL) [w3c] 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. Users can also import different set of rules to extract different ontology and service markup formats. The main process and techniques for the translation are depicted by Figure 5.

## Phase 4: Refinement of implementation code from formal specification.

The formal model resulted from phase 1 and 2 give a good understanding and a simple description of the legacy application. The tool developed in phase 3 allows us to build domain ontologies and service markup more easily. However, the specification may also be used in such a way that can lead towards a suitable Web service implementation. Generation of code from formal specification is a popular research area which has had a considerable amount work, and in which significant amount of tools and systems already exist [WD96, SCW98, AN96, RC92]. However, the refinement from formal models to Web Service-specific implementation is a relatively new research area. The details of the refinement calculus are beyond the scope of this paper and will be addressed in a separate paper.

To summarize, Section 3 presented the major challenges for migrating a legacy system to a Semantic Web Service platform. The methodology we have just presented tackles these difficulties. Firstly, since the software requirements model is derived directly from implementation code, and is represented in a rigorous way, it precisely shows the latest information about all the system functionalities and developers can understand them easily. Secondly, as the resulting requirements formal model focuses on the system's core functionalities at a high level, many implementation details are ignored. Therefore, the model is more loosely coupled compared to the implementation code, and can be more easily decomposed to services. Furthermore, by using the tool we developed, domain ontologies and service ontologies will be automatically generated. The quality of the generated ontologies can be ensured by formally validating and verifying the requirements models before ontology generation. Many existing tools can assist this *V&V* process. Finally, it is possible to get the final implementation automatically by refining the formal requirement model.

## Evaluation

The current evaluation is mainly focused on the second and third phases of the methodology. DSO National Laboratories (DSO) tried to migrate some of their existing military plan applications to Semantic Web. To evaluate our approach, firstly, an ontology about military plan has been developed directly from the existing documents and applications, mainly manually, but assisted with an information extraction (IE) engine developed by DSO [Lee02]. The ontology defines concepts in the military domain, including military organizations, specialities, geographic features, etc. For example, the class *MilitaryTask*, a sub
class of *MilitaryProcess*, is defined as follows.

The ontology also includes a set of instances, such as:

- military operations and tasks, defining their types, phases and their logical order
- military units, which are the participants of the military operations and tasks
- geographic locations, where the operations take place
- time points, for constraining the timing of the operations

At the same time, a Z model was developed on this military plan domain and Z/EVES [Saa97] applied to check the model and verify some desired properties. Z/EVES was developed at ORA Canada. It is an interactive system for composing, checking, and analyzing Z specifications. It supports the analysis of Z specifications in a number of ways: syntax and type checking, schema expansion, precondition calculation, domain checking, general theorem proving, etc.

After the verification, the automatic transformation tool is used to generate OWL and OWL-S ontologies from the Z model. Those ontologies generated from formal models have high quality in general compared with the manually developed ontologies.
We consider one of the manually developed instance ontologies: planE.owl. After being carefully studied by domain experts, at least 31 errors were identified. A brief statistics of this ontology can be found in Table 1. Of the 31 errors, 22 of them are relatively simple, and can be detected by the existing OWL reasoners [HM01]. There are 9 other errors which cannot be spotted by OWL reasoners. This is mainly because that OWL, being only a small fragment of FOL, has very limited expressive power. Many more complex properties within an application domain can not be captured by OWL, therefore such errors can not be detected by OWL reasoners. For example, one property we want to ensure is that for a given military task, its start time is less than its end time, and it is not a sub task of itself. Among the 9 such *hidden errors*, 2 are caused by military tasks having start time greater than end time; 4 are caused by military tasks that do not have an end time defined, and 3 are caused by a military unit being assigned to different tasks simultaneously. By contrast, because the formal model has been formally verified before the translation, the automatically generated ontologies using our methodology do not contain such errors. The following Z/EVES theorem tests that for a given military task, its start time is less than or equal to its end time and it is not a sub task of itself.

**theorem** MilitaryTaskTimeSubTaskTest1

$$\exists x : instances(MilitaryTask) \bullet \\ start(x) < end(x) \land \\ x \notin (sub\_val(subTaskOf))(|\{x\}|)$$

We are also planning to evaluate our complete methodology in some more complicated systems.

| Items | Numbers |
|---|---|
| Resources | 138 |
| Operations, tasks, phases | 56 |
| Units | 47 |
| Geographic areas | 35 |
| Statements (in RDF) | 592 |
| *Simple errors* | *22* |
| ***Hidden errors*** | ***9*** |

*Table 1: Statistics of the ontology planE.owl*

## Related work and conclusion

A key requirement of transitioning applications to Semantic Web Services has promoted the urgent need of systematic methodologies and tools to assist the migration process. In this paper, we have proposed a methodology that takes advantage of rigorous mathematical methods. Formal methods have been demonstrated to be beneficial in the development and maintenance of software. The automation of the process of abstracting a formal specification from program

code is aresearch goal, but unfortunately not completely realizable yet. However, by applying the tools that support the reverse engineering of software, a much clearer formal model can be learned which describes the functionalities of legacy systems. This precise and abstract specification, complemented with informal information, can facilitate understanding and re-implementation of systems. Furthermore, the well-defined syntax and semantics allows us to develop rules and tools to extract the markup information needed for a SW service, and provides an opportunity to automatically or semi-atuotmatically develop a high quality SW service implementation. The proposed methodology makes use of several tools, and in our subsequent work we intend to better integrate those tools together and develop a system which can provide a one stop solution to the problem of transitioning applications to Semantic Web Services.

Previous work on Semantic Web Services migration is scant. There are a number of MDA tools (such as ArcStyler[3], Borland Together[4]) which try to assist general application transformation by using certain reverse engineering technologies. They are all based on UML. UML provides a graphical notation which allows users to design and understand software systems more easily. However, one problem of using UML is that the semantics for some kinds of UML diagrams have not yet been completely formal defined, and we cannot guarantee the consistency between various diagrams which model different aspects of one system. Furthermore, because of the lack of precise semantics, the verification tools for UML are also limited. By using the formal methods and formal tools, as demonstrated in the paper, in our methodology can discover and correct some errors at early stage of the Semantic Web service transitioning process. Therefore, the quality of the resulting service can be ensured and the cost of the migration can be reduced as well.

Finally, from a complete different direction, researchers have also recently investigated how Semantic Web and Semantic Web services can be used to build a flexible environment for supporting, extending and integrating various formal specification languages [DSW02]. One additional benefit is that OWL and RDF query techniques can facilitate formal specification comprehension.

## Acknowledgment

This work is partially supported by the EU-funded TAO project (IST-2004-026460).

## References

[ABH+02] A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. DAML-S: Web Service Description for the Semantic Web. In *First International Semantic Web Conference (ISWC) Proceedings*, pages 348–363, 2002.

[3] www.interactive-objects.com

[4] http://www.borland.com/us/products/together/index.html

[AN96] JR Abrial and A Nikolaos. *The B-book: assigning programs to meanings*. Cambridge University Press New York, NY, USA, 1996. ISBN:0-521-49619-5.

[BBL91] Jonathan P. Bowen, Peter T. Breuer, and Kevin C. Lano. The redo project: Final report. Technical Report PRG-TR-23-91, Oxford University Computing Laboratory, UK, 1991.

[BBL93a] J. P. Bowen, P. T. Breuer, and K. C. Lano. A compendium of formal techniques for software maintenance. *IEE/BCS Software Engineering Journal*, 8(5):253–262, September 1993.

[BBL93b] Jonathan P. Bowen, Peter T. Breuer, and Kevin C. Lano. Formal specifications in software maintenance: From code to Z++ and back again. *Information and Software Technology*, 35(11/12):679–690, November/December 1993.

[BLHL] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. Scientific American, May 2001.

[BvHH+04] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Reference. Available: http://www.w3c.org/TR/owl-ref/, 2004.

[CC90] Elliot Chikofski and James Cross. Reverse engineering and design recovery: taxonomy. *IEEE Software*, 7(1):13–17, 1990.

[D. 04] D. Brickley and R.V. Guha (editors). Resource description framework (rdf) schema specification 1.0. http://www.w3.org/TR/rdf-schema/, February 2004.

[DSW02] J. S. Dong, J. Sun, and H. Wang. Semantic Web for Extending and Linking Formalisms. In L.-H. Eriksson and P. A. Lindsay, editors, *Proceedings of Formal Methods Europe: FME'02*, Copenhagen, Denmark, July 2002. Springer-Verlag.

[GC93] Gerald C. Gannod and Betty H. C. Cheng. A two-phase approach to reverse engineering using formal methods. In *Formal Methods in Programming and Their Applications*, pages 335–348, 1993.

[Hay93] I. Hayes, editor. *Specification Case Studies*. International Series in Computer Science. Prentice-Hall, 2nd edition, 1993.

[HL91] Howard Haughton and Kevin Lano. Ob jects revisited. In *Conference on Software Maintenance*, pages 152–161, Sorrento, Italy, May 1991. IEEE.

[HM01] Volker Haarslev and Ralf Moller. RACER system description. *Lecture Notes in Computer Science*, 2083:701–705, 2001.

[Hor98] I. Horrocks. The FaCT system. *Tableaux'98, Lecture Notes in Computer Science*, 1397:307–312, 1998.

[LB90] K. C. Lano and P. T. Breuer. From programs to Z specifications. In J. E. Nicholls, editor, Z User Workshop, Oxford 1989, Workshops in Computing, pages 46–70. Springer-Verlag, 1990.

[Le99] O. Lassila and R. R. Swick (editors). Resource description framework (rdf ) model and syntax specification. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, Feb, 1999.

[Lee02] C. H. Lee. Phase I Report for Plan Ontology. DSO National Labs, Singapore, 2002.

[LW90] S. Liu and N. Wilde. Identifying ob jects in a conventional procedural language: an example of data design recovery. In *Proceedings of the International Conference on Software Maintenance 1990*, pages 266–271. IEEE, IEEE Computer Society Press, 1990.

[MSZ01] S. McIlraith, T. Son, and H. Zeng. Semantic web services, 2001.

[RC92] G.-H. Bagherzadeh Rafsanjani and S. J. Colwill. From object-z to c++: A structural mapping. In *Proceedings of the Z User Workshop*, pages 166–179, London, UK, 1992. Springer-Verlag.

[Saa97] M. Saaltink. The Z/EVES system. In J. P. Bowen, M. G. Hinchey, and D. Till, editors, *ZUM'97: Z Formal Specification Notation, volume 1212 of Lecture Notes in Computer Science*, pages 72–85. Springer-Verlag, 1997.

[SCW98] S. Stepney, D. Cooper, and J. C. P. Woodcock. More powerful data refinement in Z. In J. P. Bowen, A. Fett, and M. G. Hinchey, editors, *ZUM'98: The Z Formal Specification Notation*, volume 1493 of Lect. Notes in Comput. Sci., pages 284–307. Springer-Verlag, 1998.

[SDLW01] J. Sun, J. S. Dong, J. Liu, and H. Wang. Ob ject-Z Web Environment and Projections to UML. In *WWW-10: 10th International World Wide Web Conference*, pages 725–734. ACM Press, May 2001.

[Spi89] J.M. Spivey. The Z Notation: A Reference Manual. International Series in Computer Science. Prentice-Hall, 1989.

[W3C] World Wide Web Consortium (W3C). Extensible stylesheet language (xsl). http://www.w3.org/Style/XSL.

[WD96] J. Woodcock and J. Davies. Using Z: Specification, Refinement, and Proof. Prentice-Hall International, 1996.