Model Transformation from OWL-S to BPEL Via SiTra

Behzad Bordbar[1], Gareth Howells[2], Michael Evans[1], and Athanasios Staikopoulos[1]

[1] University of Birmingham, UK
{B.Bordbar,A.Staikopoulos,M.E.Evans}@cs.bham.ac.uk
[2] University of Kent, UK
W.G.J.Howells@kent.ac.uk

Abstract. Although there are a large number of academic and industrial model transformation frameworks available, allowing specification, implementation, maintenance and documentation of model transformations which provide a rich set of functionalities, such tools are inherently complex. In particular, for a newcomer to the field of model transformation and for researchers who are only interested in experimentation and creation of prototypes, the steep learning curve is a significant hurdle. There is thus a clear scope for the creation of model transformation frameworks that are both easy to use and able to conduct complex transformations. Simple Transformer (SiTra) is a model transformation framework, which was originally designed to be a "way in" for the experienced programmer, to start using the concepts of model transformation, and for academic researchers to experiment with the creation of prototypes of implementation of their transformations. The underlying idea of SiTra is to put less focus on the specification language, maintenance and documentation aspects of transformation, by focusing on the implementation of transformations. SiTra makes use of Java for the specification of transformations. This alleviates the need to learn a new specification language or get to grips with a new tool and development environment. SiTra is equipped with a strong transformation engine to execute the transformation behind the scenes. This paper reports on a case study involving transformations from Ontology Web Language-Service (OWL-S) to Business Process Execution Language (BPEL), demonstrating that SiTra can also be used to handle complex and large transformations.

1 Introduction

Model Driven Development (MDD) [1] is an emerging technology for software development, promoting the role of models and automatic creation of code by predefined model transformations. A variant of MDD suggested by the Object Management Group (OMG) is the Model Driven Architecture (MDA) [2, 3]. MDA provides an enabling infrastructure with standard specifications facilitating the definition and implementation of model transformations between Meta Object Facility (MOF) [4] compliant languages. The application of model transformations is expected to improve the software development process in many ways, as it enhances productivity, portability, interoperability, ease of use, maintenance and reusability [3, 5, 6].

At the moment, there are many industrial [7-9] and academic [10, 11] model transformation tools available; for a detailed list refer to [12]. These tools bring enormous benefit to the developers. For example, they include repository of models for reuse. They also make use of high-level languages for defining transformation. For example, [7], [10] and [11] makes use of, scripting language JPython, ATL and Kermeta, respectively. However, model transformation frameworks are complex. For a newcomer to the field of model transformation learning a framework is a serious impediment. Simple Transformer (SiTra) [13] is a model transformation framework, which is designed to be a "way in" for experienced programmers, to start using the concepts of model transformation, and for academic researchers to experiment with the creation of prototypes of implementation of their transformations. SiTra, which is written in Java, also makes use of Java for specification of transformations. This alleviates the need to learn a new specification language or getting to grips with a new tool and development environment.

SiTra has been successfully applied to the bench mark example of [14] and is documented in [13]. In this paper, we shall further evaluate SiTra by conducting a case study involving transformations from Ontology Web Language for Services (OWL-S) [15] to Business Process Execution Language (BPEL) [16]. A copy of SiTra is available for free download at [17].

The structure of the paper is as follows: Section 2 provides an overview of the Web service and model transformation in the context of Web services. Section 3 briefly describes SiTra and its architecture. Section 4 presents the case study of transformation from OWL-S to BPEL and discusses various outcomes of the study. Finally, section 5 presents a conclusion and draws a summary.

2 Preliminaries

This section describes introductory notions used in this paper. Firstly, a short review of Web Service languages such as WSDL, BPEL and OWL-S will be presented. Secondly, a brief review of existing research on model transformation for Web Services will be given.

2.1 Web Services

Web service technology [18] promises to provide a new level of functionality on top of the existing Web infrastructure, allowing applications to share data and to benefit from the capabilities of other applications, independent of the platforms and languages used to build them. The technology aims to facilitate the composition of a number of Web services in order to create a single service with richer functionality than any of the constituent services. In order to achieve this, the creation of languages for describing services and their interactions has received considerable attention. This paper deals with three such languages; Web Service Description Language (WSDL), Business Process Execution Language (BPEL) and Ontology Web Language – Services (OWL-S).

2.1.1 Web Service Description Language

The Web Service Description Language (WSDL) [19] is an XML language for describing Web services, particularly as service interfaces. The description separates the abstracted functionality offered by a service from the concrete details of how and where that functionality is offered. Its role and purpose can be compared to that of IDLs in conventional middleware languages such as CORBA.

A WSDL file has an abstract part, which specifies information such as signatures of operations offered by the service, the messages that are exchanged between providers and requestors as input, output and fault parameters of these operations. The concrete part of a WSDL file defines the protocol bindings and location of such services [20].

A WSDL file provides all the necessary information to assess and invoke the operations of a service. However, WSDL files do not provide any additional (semantic) information indicating, for example, “what the service does” and “how it is to be employed”. To include such information, languages such as OWL-S are used to capture the semantics of the Web service. WSDL documents provide a mechanism for expressing simple behaviour. However, describing complex interactions, such as business processes, require using other languages such as BPEL or WSCI [21].

2.1.2 Business Process Execution Language

The Business Process Execution Language for Web Services (BPEL) [16] provides an XML based-language for the formal specification of business processes and business interaction protocols. A BPEL file makes use of the WSDL file of involving services. Consequently, BPEL can be seen as an extension of WSDL [19] that provides basic one-way or request-response mechanisms for the Web service inter-communication. BPEL is designed for expressing processes in detail, allowing composition and coordination of activities such as for sequential, parallel, iterative, conditional, compensational and fault execution [21]. Hence, business process expressing interaction between services can be specified elegantly.

2.1.3 Ontology Web Language for Services

The Ontology Web Language for Services (OWL-S) [15] is also an XML based language, which facilitates capability-driven description of Web Services. It supports automatic Web service discovery, invocation, composition and interoperation based on the semantic descriptions of Web services (OWL Services Coalition [15]). An OWL-S service profile allows us to specify “who provides the service” and “what function is computed by the service”. In addition, the profile contains an expandable list of service parameters, allowing the characteristics of the service to be described in detail.

OWL-S differs from other Web service languages (for example WSDL), which offers descriptions of the syntax of the messages used in accessing a service, exposing the operations and protocols it utilizes. An OWL-S service description permits the inclusion of machine-readable information, which describes the service's capabilities in terms of the function(s) it performs, the preconditions and effects of these functions and how the service relates to other Web services. These features support the representation of Web services on the (still somewhat conceptual) Semantic Web [22]. The computer-interpretable representation of a Web service that OWL-S enables provides the potential to develop software that can automatically create composite web services by selecting and composing existing services. Such software could represent savings in the time and effort expended by Web users in searching for appropriate services along with providing a method of selecting Web services which is much more efficient and effective than searching using existing engines.

In addition, OWL-S allows the definition of composite processes [15] which can be built from the basic atomic processes (grounded in WSDL) of a number of different services. Composite processes can be defined dynamically by an agent or statically as part of the description of some virtual Web service. They are realised by executing atomic processes in a structured way (in a predefined sequence, for example, or concurrently) and by passing the results of executions as inputs to other atomic processes.

2.2 Model Transformation in Web Services

Recently, application of model transformation techniques to the development of Web services has received considerable attention [23-26], among others. Bézivin et al [23] use the ATL [10] transformation language and ATLAS engine to generate Platform Specific models from UML class and EDOC models to three different target platforms namely Java, Web Services and Java Web Service Developer Pack (JWSDP). UML Activity diagrams are well suited for platform independent modeling of business processes. Transformation of such models to BPEL and WSCI are studied in [27] [25] and WSCI [28], respectively. Koehler et al [29] investigates model driven transformations based on graph-theoretic methods to define the mapping among models possessing formal semantics, which in turn are used to analyze and synthesize the business protocol specifications. Finally, [30] uses the YATL transformation language to map and apply transformations from EDOC models to Web services.

The focus of this paper is on model transformation techniques and challenges. However, another important focus for research is to develop tools for the transformation of OWL-S specifications to BPEL specifications; as these two languages provide complimentary capabilities. BPEL is well supported by the software vendors as the favorite choice for the execution of the web services. BPEL is not designed for addressing the challenges of the semantic web. On the other hand, OWL-S is not designed for execution. Mandell and McIlraith [31] provide an interesting investigation into this area, by attempting to adapt BPEL4WS for the Semantic Web. This paper presents an alternative approach by transferring OWL-S models to BPEL, so models of the semantics of the system are expressed in OWL-S the execution is conducted in BPEL, which provides better tool and support.

3 Simple Transformer (SiTra)

There are many industrial and academic case tools supporting model transformation [7-11]. It poses as a question: why to attempt introducing yet another model transformation framework? To answer this, it is crucial to notice that a model transformation consists of two major steps. The first step is to define and specify the model transformation. This is often a complex task involving significant domain knowledge and understanding of both the source and target model domains. For example, defining a model transformation from OWL-S to BPEL, not only requires understanding of both languages, but also requires an analytic approach to discover the correct mapping between the model elements. The second step is to execute the transformation. Currently, elegant execution of the specifications is still a research issue in many cases and may require significant manual intervention in order to provide a correct implementation. In a large project, it is possible to divide the specification and implementation between two different groups of people who have relevant skills. In the case of smaller groups of developers, newcomers to MDD, and budding academic researchers, the combined effort involved in becoming an expert in the two sets of skills described above is overwhelming. In particular, the steep learning curve associated with current MDD tools is an inhibitive factor in the adoption of MDD by even very experienced programmers. SiTra aims to address the above issues by proposing a simple Java library for supporting a programming approach to writing transformations, based on the following requirements:

Use of Java for writing transformations: This relinquishes the programmer from learning a new language for the specification of transformations
Minimal framework: To avoid the overhead of learning a new Java library, the presented method has a very small and simple API

3.1 Introducing SiTra

The architecture of SiTra is depicted in Fig.1. A transformation specifies how elements of the Metamodel of the source are mapped into the elements of the Metamodel of the destination. A transformation framework, creates a destination model, which is an instance of the destination metamodel, from a source model, which is an instance of the source metamodel. Because, SiTra uses Java, as depicted in the picture, Metamodels of the source/destination and the model of the source must be created in Java. These could be provided using a Java implementation of a MOF repository, or more usually by providing an implementation of the metamodel using Java classes. For smaller models these can be created manually. For larger models one of numerous existing UML to Java tools could be used.

As depicted in Fig.1, the transformation in SiTra is provided by a number of Java classes, each of which corresponds to a model transformation rule. These classes must implement the SiTra interface Rule. The SiTra class and corresponding interface Transform are used to execute the defined transformation on a particular source model. The two simple interfaces for supporting the implementation of transformation rules in Java are shown below. The Rule interfaces should be implemented for each transformation rule written. The Transformer interface is implemented by the transformation algorithm class, and is made available to the rule classes

```
interface Rule<S,T> {
  boolean check(S source);
  T build(S source, Transformer t);
  void setProperties(T target, S source, Transformer t);
}
interface Transformer {
  Object transform(Object source);
  List<Object> transformAll(List<Object> sourceObjects);
  <S,T> T transform(Class<Rule<S,T>> ruleType, S source);
  <S,T> List<T> transformAll(Class<Rule<S,T>>    ruleType,List<S> source);
}
```

3.2 Rules

A transformation problem is split up into multiple rules; the SiTra library facilitates this, using the Rule interface. A class that implements this interface should be written for each of the rules in the transformation. The methods of this interface are described as follows:

1. The implementation of the check method should return a value of true if the rule is applicable to the source object. This is particularly important if multiple rules are applicable for objects of the same type. This method is used to distinguish which of multiple rules should be applied by the transformer.
2. The build method should construct a target object that the source object is to be mapped to. A recursive chain of rules must not be invoked within this method.
3. The setProperties method is used for setting properties of the target object (attributes or links to other objects). Setting the properties is split from constructing the target so that recursive calling of rules is possible when setting properties.

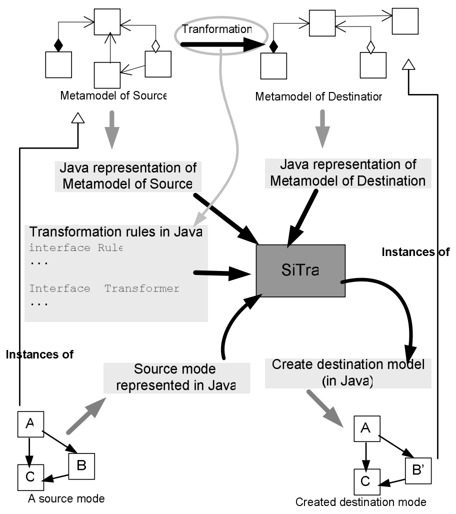

Tranformation
Metamodel of Source
Metamodel of Destination
Java representation of Metamodel of Source
Java representation of Metamodel of Destination
Transformation rules in Java
interface Rule
...
Interface Transformer
...
SiTra
Instances of
Instances of
Source mode represented in Java
Create destination model (in Java)
A
B
C
A
B'
C
A source mode
Created destination mode

Fig. 1. An outline of the SiTra framework

If it is impossible to distinguish between multiple rules using the check method, explicit rule invocation must be used to transform objects for which multiple rules apply. Objects that are derived from properties of the source object should be converted to objects for properties of the target object by calling the transform method on the transformer. However, the power transformation algorithm of SiTra manages the details of the transformation automatically. For example, it keeps the track of the objects, which are already mapped.

3.3 Transformer

To instantiate a SiTra transformation, the rule classes must be added to an instance of the SimpleTransformer class. The transformation can then be executed by calling the transform method with the root object(s) of the source model. An abstraction of the transformation algorithm is as follows:

```
FOR EACH rule
 IF rule.check(source) THEN
  IF notRecorded(source, rule) THEN
    target = rule.build(source, this)
    record(source, target, rule)
         rule.setProperties(source, this)
```

The algorithm runs through all rules in order to check which rule can be applied to a source objects. We are only interested in the source object which are not transformed yet, this is checked via the method notRecorded(). For such objects, the method build is applied which results in the creation of a target object. To ensure that a source object is not transformed more than once, the method record captures the correspondence between the source, target and rule. Finally, the method setProperties is invoked to assign further properties and attributes to the source object.

4 Case Study: Transformation from OWL-S to BPEL

In this section we shall present our case study of applying SiTra to transformation of the models from OWL-S to BPEL. We shall start by presenting metamodels of OWL-S and BPEL in the next two sections.

4.1 Metamodel of OWL-S

Fig.2 presents a metamodel of the OWL-S following [15]. We shall explain some of the model elements. An OWL-S process is a specification of the ways a client may interact with a service. A process gives a detailed perspective on how to interact with a service. Fig.2 depicts various attributes of OWL-S process. For example, a process will not execute properly unless its preconditions are true. Preconditions are logical statement representing Conditions. The attribute has Precondition specifies one of the preconditions of the service and ranges over a Precondition instance defined according to the schema in the Process ontology.

An Atomicprocess is a (process) description of a service that expects one (possibly complex) message and returns one (possibly complex) message in response. In contrast, a composite process (not depicted in Fig.2, due to space limitations) is one that maintains some state; each message the client sends advances it through the process.

OWL-S makes use of WsdlGrounding for referring to WSDL constructs. Each WsdlGrounding instance, in turn, contains a list of WsdlAtomicProcessGrounding instances. A WsdlAtomicProcessGrounding instance refers to specific elements within the WSDL specification, using the properties such as wsdlService, wsdlPort, wsdlInputMessage as depicted in Fig.2. For example, wsdlService, wsdlPort present the URI of a WSDL service (or port) that offers the given operation. For further details on OWL-S we refer the reader to [15].

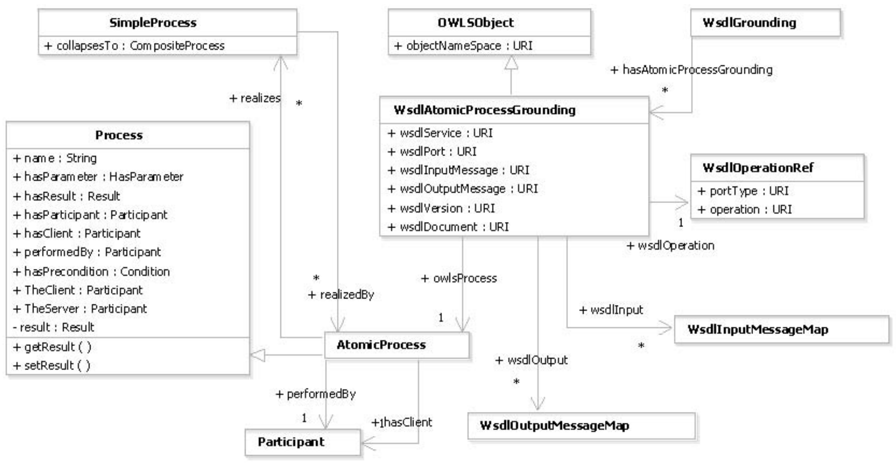


Fig. 2. A portion of OWL-S metamodel

4.2 Metamodel for BPEL

The BPEL specification can be represented by an equivalent MOF compliant metamodel, as the one depicted in Fig.3. As such, the metamodel specifies a number of

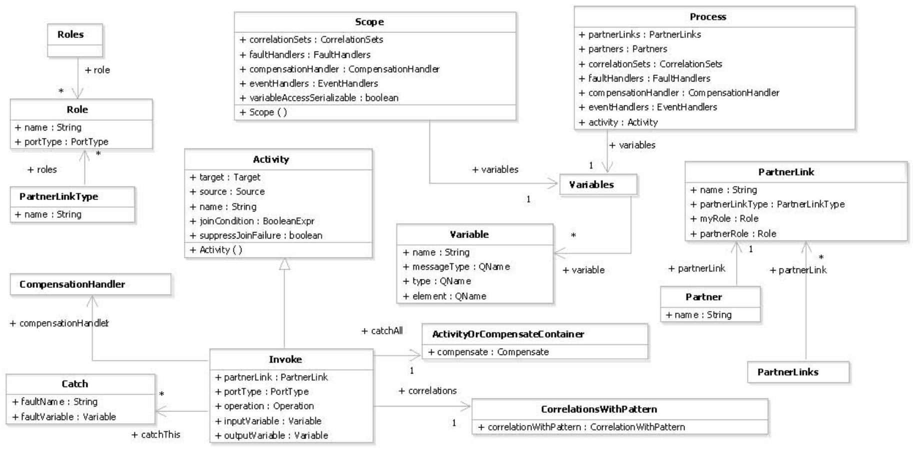

Roles
+ role
*
Role
+ name : String
+ portType : PortType
*
+ roles
PartnerLinkType
+ name : String
Scope
+ correlationSets : CorrelationSets
+ faultHandlers : FaultHandlers
+ compensationHandler : CompensationHandler
+ eventHandlers : EventHandlers
+ variableAccessSerializable : boolean
+ Scope ( )
Process
+ partnerLinks : PartnerLinks
+ partners : Partners
+ correlationSets : CorrelationSets
+ faultHandlers : FaultHandlers
+ compensationHandler : CompensationHandler
+ eventHandlers : EventHandlers
+ activity : Activity
+ variables
1
+ variables
1
Variables
Activity
+ target : Target
+ source : Source
+ name : String
+ joinCondition : BooleanExpr
+ suppressJoinFailure : boolean
+ Activity ( )
Variable
+ name : String
+ messageType : QName
+ type : QName
+ element : QName
*
+ variable
PartnerLink
+ name : String
+ partnerLinkType : PartnerLinkType
+ myRole : Role
+ partnerRole : Role
1
*
+ partnerLink
+ partnerLink
Partner
+ name : String
CompensationHandler
+ compensationHandler
+ catchAll
ActivityOrCompensateContainer
+ compensate : Compensate
Invoke
+ partnerLink : PartnerLink
+ portType : PortType
+ operation : Operation
+ inputVariable : Variable
+ outputVariable : Variable
1
+ correlations
1
CorrelationsWithPattern
+ correlationWithPattern : CorrelationWithPattern
Catch
+ faultName : String
+ faultVariable : Variable
*
+ catchThis
PartnerLinks

Fig. 3. A (partial) BPEL Metamodel

model elements that are equivalent to XML constructs, defining various activity types, which allow sequential, parallel, conditional or repetitive processing of actions. In addition it defines a number of other features, such as variables, execution context (scope) and exceptions, allowing the creation of complicated and realistic processes, performing various invocation styles and data manipulations in an algorithmic manner. For description of BPEL metamodel see [23] and [27].

4.3 Mapping of Elements

The following tables depict the correspondence between some of the model elements of OWL-S processes. For example, AtomicProcess in OWL-S, the element representing the most basic class of Web service processes, is mapped to a BPEL Process. The mapping also requires the creation of a number of other BPEL and WSDL elements so that the meaning of the output model corresponds entirely to that of the input. In addition to the main Process element, a PortType, an Operation, a PartnerLinkType, an Invoke and a Role must be created. The properties of these model elements must correspond to those of the source OWL-S AtomicProcess. From the collection of inputs belonging to an AtomicProcess, a single BPEL Input can be created, along with an associated Message and Variable. Each of the OWL-S Inputs corresponds to a single Part in the BPEL Message. OWL-S outputs can be converted to BPEL in an identical manner.

There are several examples in the table in which the OWL-S Process model element does not map to anything in BPEL or WSDL. For example, OWL-S precondition and result elements are used to incorporate semantic information regarding the change of state that occurs when the process is executed. Such notions have no equivalent in BPEL, hence it is not possible to map them to BPEL. BPEL does not support the representation of information of this nature, and thus no mapping exists for these elements.

A key set of mappings from OWL-S to BPEL involves OWL-S ControlConstruct elements. These, in general, correspond to BPEL Activity objects and are used to describe the nature in which the components of a Web service process are executed. Both OWL-S and BPEL provide similar constructs for this purpose. For example, both languages contain a Sequence element, indicating that any processes contained within should be executed strictly in order. An OWL-S Sequence maps directly to a BPEL Sequence. Some of the other ControlConstructs map to BPEL elements in a similar fashion. However, some of the OWL-S ControlConstructs have no corresponding BPEL Activity. For example, the OWL-S AnyOrder construct indicates that its component processes should all be executed, but in no particular order. BPEL has no corresponding construct. It is possible, though, to model OWL-S AnyOrder elements as BPEL Sequence elements without affecting the functionality of the output BPEL model (this may affect the efficiency of its execution, however). Therefore, the following table 1 shows the corresponding BPEL element for an OWL-S AnyOrder to be a Sequence. A similar situation arises around OWL-S Choice constructs.

Table 1. Equivenlent mapping of OWL-S Process and BPEL elements

OWL-S Process	BPEL
<Process>	<Process>
<AtomicProcess>	<Process> + <PortType> + <Operation> + <PartnerLinkType> + <Role> + <invoke> Operation + PortType names must be consistent with the created WSDL file. **Note:** require an <invoke> call within <Process> tags and Operation + PortType names must be consistent with the created WSDL file.
[<input>]* (of Atomic Process)	<input> + <message> + <variable>
<input>	<part> (of the <message> created for all inputs)
[<outputs>]* (of Atomic Process)	<output> + <message> + <variable>
<output>	<part> (of the <message> created for all outputs).
<precondition>	**Note:** This does not really map, due to complicated representation of Preconditions in OWL-S.
<result>	**Note:** This contains semantic information about the output of a process and does not map usefully to any BPEL class. The <withOutput> property will be covered by the mappings described above.
<participant>, <SimpleProcess>, <CompositeProcess>	<partner>
[<input>]* of Composite Process	<message> + <variable>
<input>	<part> (of the message created for all inputs)
[<outputs>]*of Composite Process	<message> + <variable>
<output>	<part> (of the message created for all outputs).
<ControlConstruct>	<Activity> (See below)
<sequence>	<sequence>
<Iterate>, <RepeatUntil>, <RepeatWhile>	<while>
<AnyOrder>	Model as <sequence>
<Split>	<flow>
<Choice>	Model as <switch>
<IfThenElse>	<switch>
<components> of a sequence, split... etc.	See below...

Table 1. *{Continued}*

<Binding> (as in <hasDataFrom> property) toParam, valueSource, theVar, fromProcess	<variable> (or part thereof) corresponding to that generated for the given process and parameter.
<valueSpecifier>, <SplitJoin>, <Perform>, <Produce>	-

The OWL-S Profile [15] of a Web service contains semantic information regarding what functionality the service offers, who is likely to use the service and what properties it shares with other similar services. BPEL provides no means of representing such information and therefore, there is no mapping between the OWL-S Profile and BPEL. However, some of the elements contained in the OWL-S Profile can be converted into useful human-readable information. Where possible, the transformation between OWL-S and BPEL should support the conversion into text of such elements along with their inclusion in the output BPEL file(s). The following table 2 shows examples where text conversion may be useful.

Table 2. Equivenlent mapping of OWL-S Profile and BPEL elements

OWL-S Profile	BPEL
<input>, <output>	See Process Section
<profile>, serviceName, textDescription contactInformation	Incorporate into process name(s). Include as plain text.
<precondition>, <result>, <parameter>, <ServiceParameter>, <ServiceCategory>, <ServiceClassification>, <ServiceProduct>.	-

The Grounding section [15] of an OWL-S service model contains pointers to a WSDL file in which details of how to access the real Web service processes that make up an OWL-S process are given. The mapping between the OWL-S Grounding and WSDL is, therefore, trivial, with the names of OWL-S elements corresponding directly to elements in BPEL. The following table 3 shows this mapping in full.

Table 3. Equivenlent mapping of OWL-S Grounding and WSDL/BPEL elements

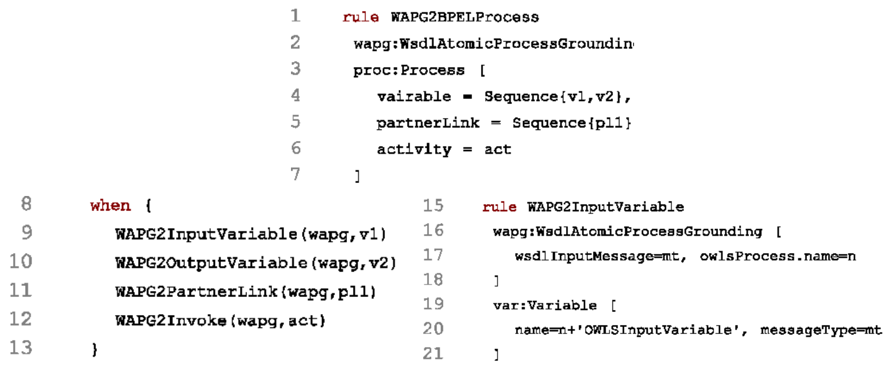
OWL-S (Grounding)	WSDL/BPEL
<owlsProcess>	Mapping to BPEL dealt with by Process Model.
<wsdlOperation>	<wsdl:Operation>
URI: wsdlPortType	<wsdl:PortType>
URI: wsdlService	<wsdl:Service>
URI: wsdlPort	<wsdl:Port>

Table 3. *{Continued}*

URI: wsdlInputMessage	<wsdl:Message>
<wsdlGrounding>, <wsdlInputMessageMap>, URI:owlsParameter, URI: xsltTransformation	-
URI: wsdlMessagePart	<wsdl:Part>
URI: wsdlVersion	(<wsdl version="…">)
URI: wsdlDocument	Name of wsdl doc/target namespace.

4.4 Model Transformation

This section will describe examples of transformation from OWL-S to BPEL. For an elaborate list, see [32]. Consider transforming the model element WsdlAtomicProcess-Grounding (for short WAPG), described briefly in section 4.1 to Business Process (for short BPELProcess). The WAPG correspond to an operation related to a given atomic process. The following snippet of code depicts a QVT-like rule for the transformation:



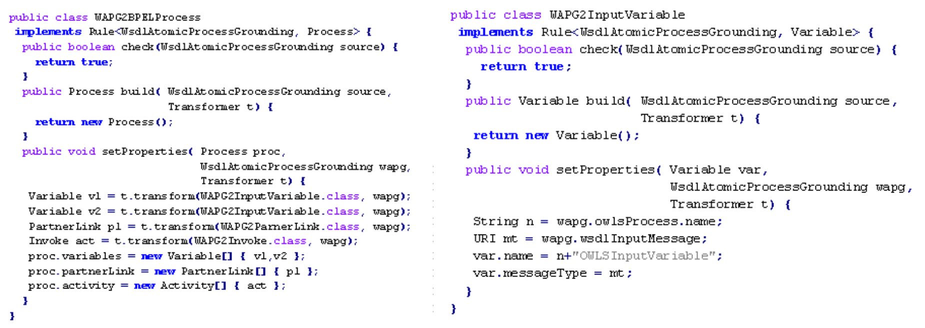
```
1    rule WAPG2BPELProcess
2     wapg:WsdlAtomicProcessGroundin
3     proc:Process [
4       vairable = Sequence{v1,v2},
5       partnerLink = Sequence{pl1}
6       activity = act
7     ]
8    when {
9      WAPG2InputVariable(wapg,v1)
10     WAPG2OutputVariable(wapg,v2)
11     WAPG2PartnerLink(wapg,pl1)
12     WAPG2Invoke(wapg,act)
13   }
```

```
15   rule WAPG2InputVariable
16    wapg:WsdlAtomicProcessGrounding [
17      wsdlInputMessage=mt, owlsProcess.name=n
18    ]
19    var:Variable [
20      name=n+'OWLSInputVariable', messageType=mt
21    ]
```

Fig. 4. QVT like rules

It can be seen that, the above snippet (Fig.4) contains two rules: The first rule, WAPG2BPELProcess, is implemented by calling four rules described in lines 9-12 within the when clause. It uses two variables v1 and v2 that correspond to InputVariable and OutputVariable accordingly of the Invoke operation of the atomic process, triggered by the PartnerLink p1. The snippet also describes the rule WAPG2InputVariable, which maps and modifies the variable's name and messageType. The equivalent SiTra code written in Java is depicted in Fig.5.

The QVT-like specification of the transformation rules map quite cleanly into Java Classes that implements the SiTra Rule interface. The build method for each rule can be seen to construct a new object of the appropriate target class. The check method in these examples simply returns a value of true as we do not need to perform any specific checks. The main work in these classes is performed within the setProperties method. This method sets the properties of the newly constructed target object according to



```
public class WAPG2BPELProcess
 implements Rule<WsdlAtomicProcessGrounding, Process> {
  public boolean check(WsdlAtomicProcessGrounding source) {
    return true;
  }
  public Process build( WsdlAtomicProcessGrounding source,
                        Transformer t) {
    return new Process();
  }
  public void setProperties( Process proc,
                             WsdlAtomicProcessGrounding wapg,
                             Transformer t) {
   Variable v1 = t.transform(WAPG2InputVariable.class, wapg);
   Variable v2 = t.transform(WAPG2InputVariable.class, wapg);
   PartnerLink pl = t.transform(WAPG2ParnerLink.class, wapg);
   Invoke act = t.transform(WAPG2Invoke.class, wapg);
   proc.variables = new Variable[] { v1,v2 };
   proc.partnerLink = new PartnerLink[] { pl };
   proc.activity = new Activity[] { act };
  }
}
```

```
public class WAPG2InputVariable
 implements Rule<WsdlAtomicProcessGrounding, Variable> {
  public boolean check( WsdlAtomicProcessGrounding source) {
    return true;
  }
  public Variable build( WsdlAtomicProcessGrounding source,
                         Transformer t) {
   return new Variable();
  }
  public void setProperties( Variable var,
                             WsdlAtomicProcessGrounding wapg,
                             Transformer t) {
   String n = wapg.owlsProcess.name;
   URI mt = wapg.wsdlInputMessage;
   var.name = n+"OWLSInputVariable";
   var.messageType = mt;
  }
}
```

Fig. 5. SiTra Code

either: properties of the source object as can be seen in the WAPG2InputVariable rule; or according to transformations of properties of the source object as illustrated by the WAPG2BPELProcess rule.

5 Discussion and Related Work

One of the important lessons learned from this case study is that the difficulty of writing transformation is independent of the choice of transformation framework. As explained in section 4.3, identifying correct mappings between elements is a challenging task, as it requires an understanding of the semantics of elements between two different domains. In this paper, as the main focus is on exploring the limitations and capabilities of SiTra, finding precise mapping of elements was of secondary importance. For example, we have decided to map the <AnyOrder> to <sequence>, forcing an order on a set of events. However, after making this decision, SiTra has helped us to write and implement the transformation in simple way. As result, the "difficulties of identifying correct transformations (whatever the language) and difficulties of writing transformation in SiTra (or other transformation frameworks) are different things". SiTra does not make the design part of creating a good set of transformation simple, it just provide and easier route to the implementation. The primary purpose of SiTra is to be simple. We strongly resisted the temptation of extending the transformer interface to overcome some limitations. We feel that this would violate our primary objective of a "simple" transformation approach. This of course has a cost, specifically that there are limitations in that we cannot tackle some of the more complex transformation problems easily. For example, a general limitations regards a situation in which there is more than one rule that should map to the same target object. There is no way to determine, using SiTra, which of the rules should construct the target object. It is necessary for the designer of the transformation to decide which rule should construct the object, to avoid such non-determinisms.

Another limitation discovered as the result of conducting the case study is regarding the recursive invocation of rules. We facilitate this by splitting the construction and setting properties of a target object. However, there is no means to enforce this, and there are potential design issues regarding situations in which some properties may need to be set in the build() method and some not (handled via setProperties() method). Identifying advantages and disadvantages of each of the two is a subject for future studies.

The graph transformation approaches [33, 34] have many merits with respect to formalism and a long history of use. However, they require a significant amount of new material to be learnt for novice users and also require significant libraries and development environments in terms of supporting framework. The source and target models are expressed using the notion of graphs, where as with SiTra, the source and target models are simple Java objects. The transformation specifications use similar concepts of rules but require a new language to be learnt for writing them, rather than the SiTra approach of using a programming language directly.

The declarative rule based approaches [35-37] suffer many of the same problems. They all require a specific model transformation specification language to be leant. Tefkat [37] and ATL [36] are both supported by a transformation engine and environment similar in concept to our Transformer implementation class (as the engine) and a Java IDE (as the environment), although in a much more heavyweight manner than SiTra.

Our Java based environment does not of course provide any specific support for debugging transformations; debugging has to be done via Java debugging tools, which are sufficient, however do make debugging a little more complex as one has to debug the rules via the internal workings of the Transformer class.

The imperative approaches such as [38] are perhaps the most similar to SiTra in terms of the style of writing a transformation rule. However, they too, all expect the transformation writer to learn a new language, and require use of a bespoke environment in which to execute the transformations.

As stated in the introduction, the SiTra library described in this paper is not intended as a replacement for a full Model Transformation Framework or as a model transformation specification language, rather it is intended as a "way in" for experienced programmers to start using the concepts of transformations rules, without the need to learn a new language, or get to grips with a new framework of tools and development environments.

Given this purpose it can be argued that a comparison between SiTra and the existing transformation languages and frameworks is not really appropriate. However, it is interesting to note what can and can't be achieved with SiTra in relation to these other approaches.

6 Conclusions

In Model Driven Development, a fundamental idea is to automatically transform models from one modelling domain to another. Consequently, providing suitable model transformation frameworks to support such transformations is of paramount importance. This paper has reported on a case study involving transformation of models in OWL-S to BPEL, via our lightweight modelling transformation framework called SiTra. SiTra uses Java for the specification of the transformation rules, significantly eliminating the need to learn any new model transformation languages or to master complex model transformation frameworks. SiTra masks the details of the execution from the user by providing a powerful execution engine to implement the transformations. The paper has presented a mapping of important model elements from OWL-S to BPEL and describes samples of transformation rules written in QVT-like language and their equivalent in SiTra. The case study demonstrates that the method adopted by SiTra is powerful enough to handle even large and complex transformations.

Some of the concepts in OWL-S have no corresponding concepts in BPEL. As a result, elements modelling such concepts cannot be mapped to BPEL. Identifying correct transformation between two modelling domain is a challenging task. Indeed, we have come to the conclusion that model transformation frameworks, including SiTra, do not make the design part of creating a good set of transformations simple. However, SiTra provides an easier route to the implementation. This is crucial, as easier routes to implementation open opportunities for better adoption of the MDD.

Acknowledgement

The authors wish to express their gratitude to David Akehurst for his assistance with this project.

References

1. Stahl, T., Volter, M.: Model Driven Software Development; technology engineering management. Wiley, Chichester (2006)
2. Frankel, D.S.: Model Driven Architecture: Applying MDA to Enterprise Computing. OMG Press (2003)
3. MDA: Model Driven Architecture, Object Management Group (2005), www.omg.org/mda/
4. MOF: Meta Object Facility (MOF) 2.0 Core Spec.: Available (2004), at http://www.omg.org
5. Kleppe, A.W., Jos & Bast, W.: MDA Explained: The Model Driven Architecture–Practice and Promise. Addison-Wesley, London, UK (2003)
6. Denno, P., Steves, M.P., Libes, D., Barkmeyer, E.J.: Model-Driven Integration Using Existing Models. In: IEEE Software, vol. 20, pp. 59–63. IEEE computer Society, Los Alamitos, CA (2003)
7. Arcstyler: Arcstyler 5.0- Interactive Objects (2005)
8. OptimalJ: Compuware Software coporation (2005)
9. XMF-Mosaic: xactium (2005), http://www.xactium.com/
10. ATLAS: ATLAS, Université de Nantes (2005)
11. kermeta: Triskell Metamodelling Kernel (2005)
12. Planetmde: Planet MDE (2005), http://www.planetmde.org
13. Akehurst, D.H., Bordbar, B., Evans, M.J., Howells, W.G.J., McDonald-Maier, K.D.: SiTra: Simple Transformations in Java. ACM/IEEE 9TH International Conference on Model Driven Engineering Languages and Systems, Vol. 4199, pp. 351–364 (2006)
14. Bezivin, J., Rumpe, B., Schurr, A., Tratt, L.: A bench mark for model tranformation, see the Call for Papers at sosym.dcs.kcl.ac.uk/events/mtip05/long_cfp.pdf. Model Transformations in Practice Workshop, part of MoDELS 2005 (2005)
15. OWL-S: OWL Services Coalition (2004), OWL-S: Semantic Markup for Web Services. (2004), http://www.daml.org/services/owl-s/1.1
16. BEA, IBM, Microsoft, SAP, A., Systems, S.: Business Process Execution Language for Web Services. Version 1.1. (2003)
17. SiTra: Simple Transformer (SiTra): an MDE tool http://www.cs.bham.ac.uk/b̃xb/SiTra.html
18. W3C: Web Services Architecture (2004)
19. Chinnici, R., Moreau, J.-J., Ryman, A., Weerawarana, S.: Web Services Description Language (WSDL) Version 2.0, W3C (2006), http://www.w3.org/TR/wsdl20/

20. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services. Springer, Berlin (2004)
21. W3C: Web Service Choreography Interface (WSCI) 1.0, W3C Note (2002)
22. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. Scientific American (2001)
23. Bézivin, J., Hammoudi, S., Lopes, D., Jouault, F.: An Experiment in Mapping Web Services to Implementation Platforms. Technical report: 04.01. LINA, University of Nantes, Nantes, France (2004)
24. Bordbar, B., Staikopoulos, A.: On Behavioural Model Transformation in Web Services. In: Wang, S., Tanaka, K., Zhou, S., Ling, T.-W., Guan, J., Yang, D.-q., Grandi, F., Mangina, E.E., Song, I.-Y., Mayr, H.C. (eds.) Conceptual Modeling for Advanced Application Domains. LNCS, vol. 3289, Springer, Heidelberg (2004)
25. Gardner, T.: UML modelling of automated business processes with a mapping to BPEL4WS. In: 17th European Conference on Object Oriented Programming (ECOOP) (2005)
26. Bordbar, B., Staikopoulos, A.: Modelling and Transfomation of Behavioural aspects of Web Services. In: 3rd Workshop in Software Model Engineering - WiSME2004, UML 2004, Lisbon, Portugal (2004)
27. Bordbar, B., Staikopoulos, A.: On Behavioural Model Transformation in Web Services. Conceptual Modelling for Advanced Application Domain (eCOMO), China, pp. 667–678 (2004)
28. Bordbar, B., Staikopoulos, A.: Modelling and transforming the behavioural aspects of web services Third Workshop in Software Model Engineering (WiSME) at UML, Portugal (2004)
29. Koehler, J., Hauser, R., Kapoor, S., Wu, F.Y., Kumaran, S.: A model-driven transformation method. In: Seventh IEEE International Enterprise Distributed Object Computing Conference, Brisbane, Australia, pp. 186–197 (2003)
30. Patrascoiu, O.: Mapping edoc to web services using yatl. Eighth IEEE International Enterprise Distributed Object Computing, pp. 289–297 (2004)
31. Mandell, D.J., McIlraith, S.A.: Adapting BPEL4WS for the semantic web: The bottom-up approach to web service interoperation. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, Springer, Heidelberg (2003)
32. SiTra: Simple Transformer (SiTra): an MDE tool (2006)
33. Konigs, A.: Model Transformations with Tripple Graph Grammars. Model Transformations in Practice Workshop at MoDELS 2005. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, Springer, Heidelberg (2005)
34. Taentzer, G., Ehrig, K., Guerra, E., Lara, J., Lengyel, L., Levendovszky, T., Prange, U., Varro, D., Varro-Gyapay, S.: Model Transformations by Graph Transformations: A Comparative Study. Model Transformations in Practice Workshop at MoDELS 2005. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, Springer, Heidelberg (2005)
35. Akehurst, D.H., Howells, W.G., McDonald-Maier, K.D.: Kent Model Transformation Language. Model Transformations in Practice Workshop, part of MoDELS 2005. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, Springer, Heidelberg (2005)
36. Jouault, F., Kurtev, I.: Transforming Models with ATL Model Transformations in Practice Workshop at MoDELS. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, Springer, Heidelberg (2005)
37. Lawley, M., Steel, J.: Practical Declarative Model Transformation With Tefkat. Model Transformations in Practice Workshop at MoDELS 2005. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, Springer, Heidelberg (2005)
38. Kalnins, A., Celms, E., Sostaks, A.: Model Transformation Approach Based on MOLA. Model Transformations in Practice Workshop at MoDELS 2005. In: Briand, L.C., Williams, C. (eds.) MoDELS 2005. LNCS, vol. 3713, Springer, Heidelberg (2005)