The Knowledge Engineering Review, Vol. 29:2, 248–264.
doi:10.1017/S0269888914000083

Semantic composition of business processes using Armstrong's Axioms

DUYGU CELIK[1] and ATILLA ELCI[2]

[1]*Computer Engineering Department, Istanbul Aydin University, Istanbul, Turkey;*
e-mail: duygucelik@msn.com;
[2]*Department of Electrical-Electronics Engineering, Aksaray University, Aksaray, Turkey;*
e-mail: atilla.elci@gmail.com

Abstract

Lack of sufficient semantic description in the content of Web services makes it difficult to find and compose suitable Web services during analysis, search, and matching processes. Semantic Web Services are Web services that have been enhanced with formal semantic description, which provides well-defined meaning. Due to insertion of semantics, meeting user demands will be made possible through logical deductions achieving resolutions automatically. We have developed an inference-based semantic business process composition agent (SCA) that employs inference techniques. The semantic composition agent system is responsible for the synthesis of new services from existing ones in a semi-automatic fashion. SCA System composes available Web Ontology Language for Web services atomic processes utilizing Revised Armstrong's Axioms (RAAs) in inferring functional dependencies. RAAs are embedded in the knowledge base ontologies of SCA System. Experiments show that the proposed SCA System produces process sequences as a composition plan that satisfies user's requirement for a complex task. The novelty of the SCA System is that for the first time Armstrong's Axioms are revised and used for semantic-based planning and inferencing of Web services.

1 Introduction

With contributions from Semantic Web technology, Web services are enhanced using rich description languages such as Web Ontology Language (OWL, 2004). Semantic Web Services (SWSs, McIlraith *et al.*, 2001) are Web services that have been enhanced with formal semantic descriptions in Web Ontology Language for Web services (OWL-S, 2004). OWL-S is an OWL-based Web services ontology providing Web service suppliers with a core set of markup languages that is designed for describing properties, functions, and access information for Web services in a computer-interpretable form. All structural information of Web services can be embedded through OWL-S ontology. It includes three complimentary models serving this purpose: ***Profile***, ***Process***, and ***Grounding***. ***Profile*** model describes what function a service provides or operates. While ***process*** model is a functional description of how a service works, procedure ends with ***grounding*** model describing how to access the service.

Inputs, ***Outputs***, ***Preconditions***, and ***Effects (I|O|P|E)*** are concepts of an atomic process in the OWL-S process model. ***Inputs*** and ***Outputs*** specify data transformation produced by the process. ***Inputs*** describe the information that is requested from a user/agent before executing a process. ***Outputs*** describe the information that a process produces after its execution. ***Preconditions*** are conditions that are imposed upon *Inputs* for a process prior to its invocation. An ***Effect*** can be defined as a proposition that will become true when the execution is completed.

In this article, emphasis is on Web services composition utilizing atomic business processes. Web services composition addresses a situation where a client's request cannot be met by a single available service; it can, however, be satisfied by suitably combining multiple existing services. Therefore, we concentrated on the process model of the OWL-S ontology for semantic annotation. The reason is that it provides good support for describing data and control constructs of individual Web services, with clear semantics for describing profile, process, and grounding models. Decomposition of composite processes lies outside the scope of the present study. It is assumed that all composite processes of SWSs have been already decomposed into atomic processes before planning task of the composition can be initiated.

This paper proposes a composition model involving an *inference-based semantic business process composition agent* (*SCA*) for verifying compatibility and composition of OWL-S atomic business processes. The SCA System performs composition task in order to sequence execution of business processes in such an order that accomplishes a client's complex process requirement. Framework of the SCA System was designed; a prototype was implemented and tested on a corpus of Semantic Web Services in order to demonstrate it.

The rest of this paper is organized as follows: Section 2 introduces the proposed SCA System. Section 3 investigates the Ontology and Rule Knowledge Bases used in SCA. In Section 4, OWL-S process model is introduced in order to gather useful metadata from descriptions of SWSs. While Section 5 presents the sample SWSs corpus used, Section 6 takes up the *Revised Armstrong's Axioms (RAAs)* in more detail identifying how we obtained them. Section 7 describes design aspects of process inferencing through the *RAAs* and case studies. In Section 8, the *Process Equivalency Task* is described and discussed which concludes the composition task. Sections 9 and 10 are dedicated to related works and conclusions, respectively.

2 The inference-based semantic business process composition agent

The proposed SCA System has five main parts, which are as follows: ***translator***, ***planner***, ***inference engine***, ***execution engine***, and ***monitoring agent***. The system has three ontology knowledge bases ***(Tasks-TKBO***, ***Concepts (or Domain-CKBO)*** and ***Processes-PKBO)***, and one ***Rule Knowledge Base (RKB)*** that contains the ***Revised Armstrong's Axioms***.

SCA composes OWL-S-based atomic processes using a novel planning method through the *RAAs*. The *Armstrong's Axioms (AAs*; *or more precisely, Armstrong's inference rules*) are a set of axioms that are used to infer functional dependencies on a relational database. They were developed by William W. Armstrong in his paper titled *Dependency Structures of Data Base Relationships* (Armstrong, 1974). We revised AAs in order to shorten inference chains while preserving the integrity to derive functional dependencies of the processes, and used in this work during the inferencing and planning phase. The novelty of the SCA System is that for the first time such a study revises the AAs and employs them in semantic-based planning and inferencing of Web services. This article proposes embedding the AAs in ontologies for finding appropriate composition plans of OWL-S-based atomic processes. The architecture of the SCA System is depicted in Figure 1; main parts of SCA are introduced below:

- The ***Translator*** performs a parsing task of knowledge bases namely, translating all atomic tasks in Task Knowledge Base Ontology (TKBO), all atomic processes of SWSs (candidates) in Process Knowledge Base Ontologies (PKBO) and goal process of client into I/O/P/E form. The translator utilizes the Tasks, Concepts and Processes ontology KBs, and details translation from OWL-S process model to I/O/P/E form. According to I/O/P/E modeling, a given set of task instances in the TKBO and atomic processes in the PKBO (e.g. P_i) are converted into the form $P_i \equiv I_i \rightarrow O_i$, where its inputs are $I_i \equiv I_{i1} \wedge I_{i2} \wedge \ldots \wedge I_{ik}$ and its outputs are $O_i \equiv O_{i1} \wedge O_{i2} \wedge \ldots \wedge O_{im}$. The logical expression of $P_i \equiv I_i \rightarrow O_i$ determines that O_i is obtainable only if I_i is available. The linear implication operator ('$\rightarrow$') asserts that, I_i is consumed and O_i is produced. Domain-related atomic tasks in TKBO, candidate SWSs, and goal process in I/O/P/E form are passed on to the planner.

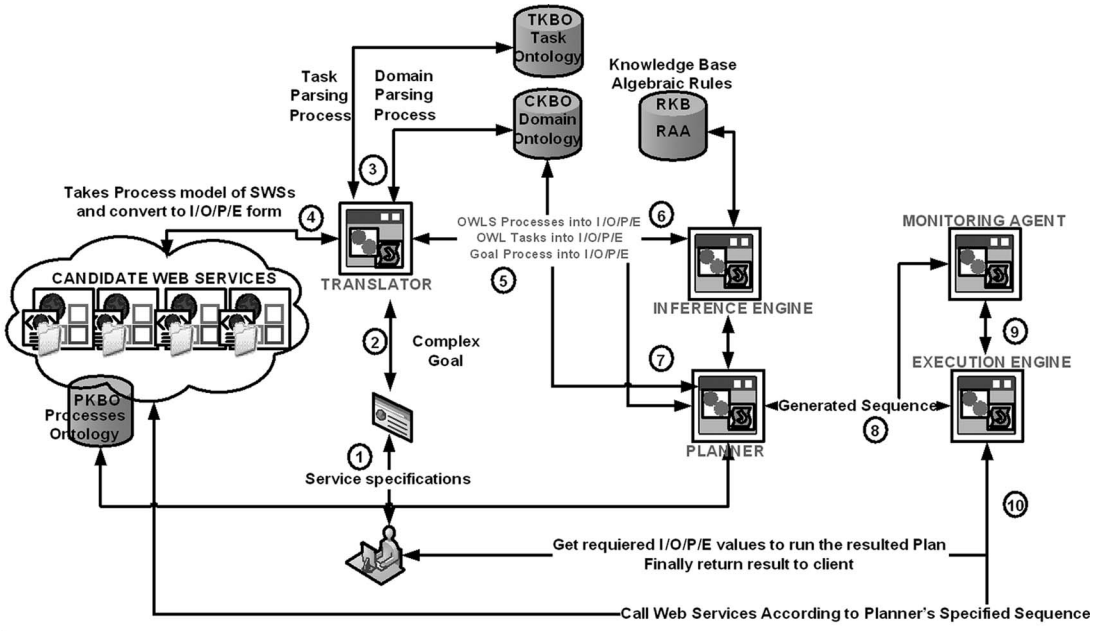

Figure 1 Architecture of the SCA System. SCA=semantic business process composition agent.

- The ***Planner*** is in the heart of the SCA System. It tries to find an execution sequence of processes using the predefined tasks (TKBO) and candidate SWSs (PKBO) while satisfying the requested complex goal process. In matching parameters, when the planner needs to find out similar concepts, it calls on the inference engine. The ***Inference Engine (IE)*** performs inferencing on the processes of candidate SWSs using the *RAAs*. A suitable RAA rule from the RKB is picked and then applied to a pair of task instances (or processes). If a new complex process is produced, it is then sent to the planner for checking its suitability. The inference engine and the planner are thus coordinated to work in alternating sequence in each iteration. In summary, there is a complex goal process $G \equiv I_G \rightarrow O_G$, with $I_G \equiv I_{G1} \wedge I_{G2} \wedge \ldots \wedge I_{Gj}$ inputs and $O_G \equiv O_{G1} \wedge O_{G2} \wedge \ldots \wedge O_{Gh}$ outputs. Given a path, P, is planned by SCA, the question is whether P is canonically implied by G ($P \models G$) or not. To resolve this question, we used a process derivation task that employs the RAAs inference rules. Inferencing using the RAAs is taken up in Section 6.
- The ***Execution Engine (EE)*** executes the planned sequence of OWL-S-based atomic processes. The sequence defined by the Planner and IE must meet the client's goal. In this paper, it is assumed that all OWL-S-based atomic processes in the planned sequence are implemented as individual SWSs and the executions of these atomic processes are triggered by the invocations of the SWSs through ***Grounding*** of OWL-S.
- Finally, the ***Monitoring Agent (MA)*** monitors proper execution of the atomic processes of the composite process during the execution stage. In the last two steps, the system initiates the execution of all the components/processes in the defined plan path. After the execution, the SCA serves the result to the client.

3 Knowledge base ontologies of the SCA system

Semantic Web is an extension of the current Web where information has a well-defined meaning. This better enables cooperation between computers and people (Lee *et al.*, 2001). Ontology, that is, the conceptual language of the Semantic Web, is a specification of a conceptualization of a knowledge domain. It is a controlled vocabulary that formally describes objects and their relationships. As mentioned above, the SCA uses three ontology knowledge bases, namely, ***Tasks***, ***Concepts (or Domain)***, and ***Processes*** knowledge bases. Effective detailing of the knowledge bases will help understanding mechanics of the SCA System. They are introduced below.

Table 1 Beginning part of the TKBO

```
<!–Datatype Properties for I/O/P/E Parameters of Tasks –>
1  <owl:DatatypeProperty rdf:ID = "Input_Parameter"/>
2  <owl:DatatypeProperty rdf:ID = "Output_Parameter"/>
3  <owl:DatatypeProperty rdf:ID = "Effect_Parameter"/>
4  <owl:DatatypeProperty rdf:ID = "Precondition_Parameter"/>
5  <owl:Class rdf:ID = "Tasks" />
6  <owl:Class rdf:ID = "e_Commerce_Service">
7     <rdfs:subClassOf rdf:resource = "#Tasks"/>
8  </owl:Class>
9  <owl:Class rdf:ID = "Information_Service">
10     <rdfs:subClassOf rdf:resource = "#e_Commerce_Service"/>
11 </owl:Class>
12 <owl:Class rdf:ID = "Buying_Service">
13    <rdfs:subClassOf rdf:resource = "#e_Commerce_Service"/>
14 </owl:Class>
15 <owl:Class rdf:ID = "Selling_Service">
16    <rdfs:subClassOf rdf:resource = "#e_Commerce_Service"/>
17 </owl:Class>
18 <owl:Class rdf:ID = "Book_Buying">
19    <rdfs:subClassOf rdf:resource = "#Buying_Service"/>
20       <Input_Parameter rdf:datatype = "&Concepts;ISBN">ISBN</Input_Parameter>
21       <Output_Parameter rdf:datatype = "&Concepts;Price">PriceOfBook</Output_Parameter>
22       <Precondition_Parameter rdf:datatype = "&xsd;anyURI">BookInStock</
 Precondition_Parameter>
23 </owl:Class>
```

TKBO=Task Knowledge Base Ontology.

3.1 Task Knowledge Base Ontology

TKBO is a common ontology that contains capability corresponding to functionality provided by Web services. TKBO provides an extensive ontology of functions or tasks. Using TKBO, Web services may be defined as instances of classes that represent their capability based on OWL-S descriptions on their service site. Furthermore, SCA System may use the TKBO during its stages for discovery, composition, and invocation of services. This is necessary in order to distinguish processes according to their functional semantics since the services may have the same inputs and outputs but have different functional semantics. The following ontology snippet presents an excerpt from the TKBO model. TKBO is used by a semantic-based SCA System for matching service capability during the inferencing and planning stage. In the TKBO, semantic task contexts of task instances are embedded by using the OWL semantic tags such as <***owl:class***>, <***rdfs:subClassOf***>, <***owl:ObjectProperty***>, and <***owl:DatatypeProperty***>. For example, the semantic task contexts in Table 1 are used to explain that the ***Tasks***, ***e_Commerce_Service***, ***Information_Service***, ***Buying_Service***, ***Selling_Service***, and ***Book_Buying*** are as <***owl:class***> classes, which keeps semantic-based descriptions for instances of tasks in e-commerce area. The line 5 declares that there is a class named ***Tasks*** using the tag <***owl:class***>, and the line 6 describes that there is another class named ***e_Commerce_Service***, which is a kind of ***Tasks*** using the tag <***rdfs:subClassOf***> at line 7. Other classes of the other tasks instances are described similarly. The lines 1–4 present defined <***owl: DatatypeProperty***> properties ***Input_Parameter***, ***Output_Parameter***, ***Precondition_Parameter***, and ***Effect_Parameter***. The lines 18–23 present a ***Book_Buying*** task having an ***Input_Parameter*** (at line 20) that specifies the association between the ***Book_Buying*** task and the ***Input_Parameter*** by <***owl:DatatypeProperty***>. The meaning of the association is that the ***Book_Buying*** task contains only one *input*, which is ***ISBN***. All other *output, precondition* and *effect* parameters of the ***Book_Buying*** task are described in the same format (lines 21–22; and there is no *effect* parameter). In this manner, various meanings related to the task can be described in the semantic task contexts. Details of inferencing and planning using <***owl:ObjectProperty***> are presented in Section 7.

3.2 Concepts Knowledge Base Ontology

CKBO is a rich ontology knowledge base that contains concepts of domains (of interest). The ontology keeps the classes/concepts of interested domain(s). The concepts in CKBO provide connection between I/O/P/E parameters of declared task instances in TKBO and those of OWL-S-based atomic processes in PKBO. The data set of the OWL-S atomic processes (in PKBO) and concepts ontology file (CKBO) in a widely used test collection can be found on SemWebCentral Website[1].

3.3 Process Knowledge Base Ontologies

PKBO is a repository that contains the OWL-S-based atomic processes of the candidate SWSs in the domain of interest. As mentioned above, we used a test collection that is extensively used in the Semantic Web research studies. It is very possible that a semantic based discovery agent can find suitable SWSs related to client's request. Discovery of SWSs possibly required also for composing a sequence of processes in order to meet client's request is discussed in our previous work (Celik & Elci, 2008). Here it is assumed that the discovery task would have been performed before initiating the composition task of the SCA. Thus, the system will be ready for a composition task since the discovery task had already returned a set of OWL-S atomic processes of related SWSs (called the set of candidate SWSs). In SCA, OWL-S files of atomic processes of candidate SWSs are collected in the PKBO.

3.4 Rule Knowledge Base

RKB keeps the predetermined inference rules, namely *RAAs*, a set of axioms used to infer new processes from a set of available ones.

4 Semantics in the OWL-S process model

In this section, we introduce semantic descriptions of Web services through OWL-S. An OWL-S Profile characterizes two aspects of service functionality: information transformation, which is represented by *Inputs* and *Outputs*, and state change, which is produced by an execution of a service.

```
1  <service:Service rdf:ID="BookPrice">
2  </service:Service>
3  <process:AtomicProcess rdf:ID="BookPrice-Process">
4      <process:hasInput rdf:resource="#BookName"/>
5      <process:hasOutput rdf:resource="#PriceOfBook"/>
6      <process:hasPrecondition>
7        <expr:SWRL-Condition rdf:ID="BookInStock">
8          <expr:expressionLanguage rdf:resource="http://www.daml.org/.../Expression.
owl#SWRL"/>
9            <expr:expressionBody rdf:parseType="Literal">
10         </expr:SWRL-Condition>
11       </process:hasPrecondition>
12 </process:AtomicProcess>
```

For example, to complete a sale, a book-selling service may require the name of a book as input, and the fact that the book is in stock (the precondition). The result of the sale encompasses production of a receipt that confirms the proper execution of the transaction, transfer of ownership, shipment of the book, and reduction of stock (as effect). The above given OWL-S fragment exemplifies a **BookPrice** SWS[2] that contains an atomic ***BookPrice-Process***. The process has one input "***BookName***", output "***PriceOfBook***", and one precondition "***BookInStock***" (at lines 4, 5 and 7).

[1] http://projects.semwebcentral.org

[2] http://www.w3.org/Submission/OWL-S/

Let us take up a **BookSearch** SWS with a single atomic process called ***BookSearch-Process***. The atomic service has only one input "***BookTitle***" and one output "***Book***" of parameter type *Concepts.owl#**Title*** and *Concepts.owl#**Book***, respectively. Here we give snippets of code for OWL-S of sample **BookSearch** SWS[3].

```
<service:Service rdf:ID="BookSearch">
  <process:AtomicProcess rdf:ID="BookSearch-Process">
      <process:hasInput rdf:resource="#BookTitle"/>
      <process:hasOutput rdf:resource="#Book"/>
  </process:AtomicProcess>
  <process:Input rdf:ID="BookTitle">
    <process:parameterType rdf:datatype="http://.ontology/Concepts.owl#Title"/>
    <rdfs:label>Book Title</rdfs:label>
  </process:Input>
</service:Service>
```

Each ***process:parameterType*** has a type specified using an URI. The type can be defined by the service provider as either a class/concept or a specific *xmlDatatype*, and the parameter must be associated with a concept in the domain ontology of the service. Let's consider the OWL-S declarations of inputs '***BookTitle***' of ***BookSearch-Process*** and '***BookName***' of ***BookPrice-Process*** as given below.

```
<service:Service rdf:ID="BookPrice">
<process:AtomicProcess rdf:ID="BookPrice-Process">
  <process:Input rdf: ID="BookName">
    <process:parameterType rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </process:Input>
</service:Service>
```

Clearly these inputs have the same meaning, but they are declared differently by their providers. Furthermore, the ***process:parameterType*** property of the first is defined as *Concepts.owl#**Title*** and of the second as *XMLSchema#**string***. A service with the ***process:parameterType*** defined as string may be composed with any other service as any service can return a ***string*** as an output. This situation would make proper composition more difficult due to the availability of unrelated services because of inadequate semantic description. Unfortunately, the use of standard *XMLSchema* data types such as *string, integer, double*, etc. in the properties of ***process:parameterType*** can lead to missed ***semantic matching of processes*** in the search of suitable processes of candidate SWSs during the constitution of the required composition plan. During planning phase, a semantic search agent will not be able to notice the equivalency of those processes from either ***process:parameterType*** property or ***rdf:ID*** *(input-output)* concepts. To solve this problem, we used the *Concepts.owl (CKBO)*. Each parameter type of task instances (in TKBO) or processes (in the PKBO) indicates a class in the *Concepts.owl* instead of using *XMLSchema* data types.

5 Sample SWSs corpus used

The data set of the OWL-S-based atomic processes (in PKBO) and an ontology of concepts (CKBO) used in this paper can be found on SemWebCentral Website. The most commonly used test collection SWS-TC includes 241 OWL-S-based SWSs. This test collection is selected since the average quality of the descriptions is somewhat better than other test collections. The OWL-S descriptions of 241 different SWSs in the SWS-TC are analyzed in the Translator stage of the SCA, and some ontological data were extracted such as the *Effects (I/O/P/E)*, *parameter name* and *parameter type (as an ontological reference)*.

[3] http://projects.semwebcentral.org/projects/sws-tc/

Table 2 Seven atomic processes from the SWS-TC 1.1 with I/O/P/E parameters

SWS Name	Input	Output	Precondition	Effect
AmazonBookPrice	BookInformation<#Book>	BookPriceValue<#Price>	NULL	NULL
CurrencyConvertor	InputAmount<#Price>, SourceCurrency<#Currency>, DestinationCurrency<#Currency>	OutputAmount<#Price>	NULL	NULL
Getbookprice	BookInfo<#Book>	BookPrice<#Price>	NULL	NULL
BookInStock	BookInfo<#Book>	ExistsInStock<#true-false>	NULL	NULL
BookPrice	Book<#Book>	PriceOfBook<#Price> ResultCurrency<#Currency>	BookInStock <#true-false>	NULL
FindBookStore	ISBN<#ISBN>	Bookstore<#Store>, BookPrice<#Price>	NULL	NULL
BookSearch	BookTitle<#Title>	Book<#Book>	NULL	NULL

Table 2 depicts seven atomic processes of OWL-S-based SWSs from the SWS-TC 1.1 with the information needed for the scope of this section. All of them have input and output concepts described in the form "***rdf:ID***<***#process:parameterType***>" and only one service contains a Precondition; no service has Effect. A possible scenario is that a user wants to find the price of a book by giving its title, and expects the price information in a specific currency such as the United States Dollars (USD). Assume that the goal task (G) is required by the client $(G \equiv (BookTitle[Text] \| InputCurrency[Currency] \| DestinationCurrency[Currency]) \rightarrow BookPrice[Price])$ where the '***InputCurrency***' is specified as Euro and '***DestinationCurrency***' is specified as USD by the user. We will continue with this scenario in the next section to show how RAAs are applied on OWL-S atomic processes.

6 Revised Armstrong's Axioms

Armstrong's Inference Rules (also referred to as *Armstrong's Axioms*) are a set of axioms used to infer functional dependencies (FDs) on a relational database. They were developed by William W. Armstrong in his paper titled *Dependency Structures of Data Base Relationships* (Armstrong, 1974). An inference axiom is a rule, which states that if a relation satisfies certain FDs, it must also satisfy certain other FDs. There are seven axioms that are ***Reflexivity***, ***Augmentation***, ***Transitivity***, ***Pseudo Transitivity***, ***Additivity***, ***Accumulation***, and ***Projectivity*** as shown in Table 3. Five of them, namely, ***Transitivity***, ***Pseudo Transitivity***, ***Additivity***, ***Projectivity***, and ***Accumulation*** are applied exactly in the same fashion during the constitution of a composition plan by the ***Planner*** and ***IE*** of SCA. We tested and found them suitable for deriving the required process sequences for a composition. On the other hand, since the ***Reflexivity*** and ***Augmentation*** axioms applied in sequence performs like factorization; we combined them together in order to form a shorthand ***Pseudo Factorization*** axiom. By using these two rules separately by Planner of SCA System, the system is likely to generate many unsuitable derivative composition plans; perhaps even end up doing infinite iterations. The combined version for these two rules will not only eliminate unsuitable derivations, it will also reduce the complexity of composition. In addition, we renamed the ***Projectivity*** axiom as ***Dissection*** for the latter better expresses the intended operation.

Table 3 displays AAs in the first column, corresponding RAAs in the second column. Let us assume that X, Y, Z, W, and T are classes/concepts of ***Input/Output*** parameters of an available process set and that the right arrow is the linear implication operator. Using example $X \| Y \rightarrow Z$ in the Table 3, we can understand that the concepts X and Y are taken as concurrent inputs (i.e. $X \| Y$) while execution of the process, thus their order is not important. The outcome is the concept Z after execution.

An important point of the revision is that, while combining ***Reflexivity*** and ***Augmentation*** axioms, we did not alter correctness and completeness of *AAs*. Therefore, the RAAs are equivalent transformations of AAs. A composition plan is associated with a control flow and a data flow of processes in the plan.

Table 3 Armstrong's Axioms and Revised Armstrong's Axioms

Armstrong's Axioms	Revised Armstrong's Axioms
1. **Reflexivity**: A set of attributes X determines a subset Y of itself: $X \rightarrow Y$ if $Y \subseteq X$.	1. **Pseudo factorization**: If $\{(X \| Y \rightarrow Z), (T \| Z \rightarrow W)\}$ and if $(Z \subset T \| Z$ and $T \equiv X$ or $T \equiv Y)$ then $X \| Y \rightarrow W$.
2. **Augmentation**: It allows enlarging the left-side of a FD or both side conventionally with one or more attributes. Formally, if $X \rightarrow Y$ then $X \| Z \rightarrow Y \| Z$ for any Z.	
3. **Transitivity**: If we have functions $f : X \rightarrow Y$ and $g : Y \rightarrow Z$ then we have a function $g \text{ O } f : X \rightarrow Z$, where $(g \text{ O } f)(x) = g(f(x))$. Formally, If $\{(X \rightarrow Y)$ and $(Y \rightarrow Z)\}$, then $X \rightarrow Z$.	2. **Transitivity**: If $\{(X \rightarrow Y)$ and $(Y \rightarrow Z)\}$ then $(X \rightarrow Z)$.
4. **Pseudo transitivity**: is a generalization of Transitivity. It is requires the entire right hand side of a FD appears as attribute(s) of the determinant of another FD. Formally, if $\{(X \rightarrow Y)$ and $(Y \| Z \rightarrow W)\}$ then $(X \| Z \rightarrow W)$.	3. **Pseudo transitivity**: If $\{(X \rightarrow Y)$ and $(Y \| Z \rightarrow W)\}$ then $(X \| Z \rightarrow W)$.
5. **Additivity**: If there are two FDs with the same determinant on the left, it is possible to form a new FD that preserves the determinant and has as its right-hand side the union of the right-hand sides of the two FDs. Formally, if $\{(X \rightarrow Y)$ and $(X \rightarrow Z)\}$ then $(X \rightarrow Y \| Z)$.	4. **Additivity**: If $\{(X \rightarrow Y)$ and $(X \rightarrow Z)\}$ then $(X \rightarrow Y \| Z)$.
6. **Accumulation**: If there are two FDs with the complementary determinant, it is possible to form a new FD that preserves the determinant X and forms its right-hand side as the union of the right-hand sides of the two FDs except the complementary determinant Z. Formally, if $\{(X \rightarrow Y \| Z)$ and $(Z \rightarrow C \| W)\}$ then $(X \rightarrow Y \| C \| W)$.	5. **Accumulation**: If $\{(X \rightarrow Y \| Z)$ and $(Z \rightarrow T \| W)\}$ then $(X \rightarrow Y \| T \| W)$.
7. **Projectivity**: If X determines Y and Z, then X determines Y, therefore it is possible to break each functional dependency $X \rightarrow Y$ down to $X \rightarrow Ai$ for $i = 1..n$ where $Y = \{A1, \ldots An\}$. Formally, if $(X \rightarrow Y \| Z)$ then $X \rightarrow Y$ and $X \rightarrow Z$.	6. **Dissection**: If $(X \rightarrow Y \| Z)$ then $X \rightarrow Y$ and $X \rightarrow Z$.

Table 4 Revised Armstrong's Axioms in the Tasks.owl (TKBO)

```
<!– Object Properties of Tasks.Owl–>
1  <owl:ObjectProperty rdf:ID = "hasTransitivity">
2    <rdf:type rdf:resource = "&owl;TransitiveProperty"/>
3      <rdfs:domain rdf:resource = "#Tasks"/>
4      <rdfs:range rdf:resource = "#Tasks"/>
5  </owl:ObjectProperty>
6  <owl:ObjectProperty rdf:ID = "hasPseudoFactorization">
7    <rdf:type rdf:resource = "&owl;TransitiveProperty"/>
8      <rdfs:domain rdf:resource = "#Tasks"/>
9      <rdfs:range rdf:resource = "#Tasks"/>
10 </owl:ObjectProperty>
11 <owl:ObjectProperty rdf:ID = "hasPseudoTransitivity">
12   <rdf:type rdf:resource = "&owl;TransitiveProperty"/>
13     <rdfs:domain rdf:resource = "#Tasks" />
14     <rdfs:range rdf:resource = "#Tasks" />
15 </owl:ObjectProperty>
16   <owl:ObjectProperty rdf:ID = "hasAdditivity">
17   <rdf:type rdf:resource = "&owl;UnionProperty"/>
18     <rdfs:domain rdf:resource = "#Tasks"/>
19     <rdfs:range rdf:resource = "#Tasks"/>
20 </owl:ObjectProperty>
21 <owl:ObjectProperty rdf:ID = "hasAccumulation">
22   <rdf:type rdf:resource = "&owl;TransitiveProperty"/>
23     <rdfs:domain rdf:resource = "#Tasks"/>
24     <rdfs:range rdf:resource = "#Tasks"/>
25 </owl:ObjectProperty>
26 <owl:ObjectProperty rdf:ID = "hasDissection">
27     <rdfs:domain rdf:resource = "#Tasks"/>
28     <rdfs:range rdf:resource = "#Tasks"/>
29 </owl:ObjectProperty>
```

A control flow addresses the task dependencies among a set of atomic processes. Here, the order of atomic processes is a crucial point. A pair of processes is called combinative *if the output of one is equal to the input requirements of the other (transitive), or, if the input of one is equal to the input requirements of the other (additive)*. RAAs are applicable in structuring a control flow (as a workflow of atomic processes) among candidate processes. If we apply ***Transitivity*** then ***Pseudo Transitivity*** rules to the ***BookSearch.owl***, ***BookPrice.owl*** and ***CurrencyConvertor.owl*** in Table 2, the possible chain is **Pseudo[Transitivity[*BookSearch*○*BookPrice*]○*CurrencyConvertor*]]**. Additionally, the expected new process (named as P) after tracing of Planner and IE actions is **P ≡ [Title[Concepts.owl#Text]‖DestinationCurrency[Concepts.owl#Currency] → OutputAmount[Concepts.owl#Price]]** that results the P is canonically implied by G ($P \models G$). Inferencing is taken up next.

7 Inferencing in the SCA System

Semantic task contexts are supported by the OWL formal semantics. Therefore, reasoning can be performed based on the semantically described task contexts. In order to do this, we employ the Task ontology (Tasks.owl-TKBO) for inference purpose. In this section, we present the details of inference through TKBO of the SCA System. OWL formal semantics can be used to define RAAs as object property characteristics to be used during the inference and planning stage. Thus ***hasTransitivity***, ***hasPseudoFactorization***, ***hasPseudoTransitivity***, ***hasAdditivity***, ***hasAccumulation*** and ***hasDissection*** are defined as <***owl:ObjectProperty***>. The RAAs rules are inserted into the Tasks ontology (TKBO) in order to describe task dependencies as displayed in the lines 1, 6, 11, 16, 21, and 26 in Table 4.

OWL formal semantics provide many property characteristics such as ***transitive***, ***symmetric***, ***functional***, ***inverseOf***, ***inverseFunctional***, etc. For instance, if a property P is specified as transitive then for any X, Y, and Z: **P(X,Y) and P(Y,Z) implies P(X,Z)**. Kang *et al*. (2009) used the property characteristics during the reasoning of business process knowledge to provide an order among some semantic business processes for monitoring purpose. However, the SCA uses the*RAAs* to describe task dependencies semantically as a kind of OWL object property. In Table 4, the defined transitive-based properties (as shown in lines 2, 7, 12, and 22) are suitable to reason about the execution order of several tasks while planning. Additionally, union-based properties (as shown in the line 17) could be also suitable for additive-based operations among a set of tasks.

To understand better the depth of inferencing in the SCA System, one will have to consider another similar example of Online Book Store domain. Four different task instances from TKBO: ***BookISBN-Task***, ***BookPrice-Task***, ***Currency_Converter-Task***, and ***BookSearch-Task*** are given in Table 5. The object property ***hasTransitivity*** is specified as transitive (lines 1 and 2 in Table 4). In addition, if the ***BookISBN-Task*** is transitive with ***BookPrice-Task*** then ***BookISBN-Task*** has to be

Table 5 Four tasks of the Online Book Store Domain in the Tasks.owl (TKBO)

```
1    <owl:Class rdf:ID = "BookISBN-Task">
2    <rdfs:subClassOf rdf:resource = "#Information_Service"/>
3    <Input_Parameter rdf:datatype = "&Concepts;Title">Book Title</Input_Parameter>
4    <Output_Parameter rdf:datatype = "&Concepts;ISBN">ISBN</Output_Parameter>
5    <hasTransitivity rdf:resource = "&Tasks;BookPrice-Task"/>
6    <hasAdditivity rdf:resource = "&Tasks;BookSearch-Task"/>
7  </owl:Class>
8  <owl:Class rdf:ID = "BookPrice-Task">
9    <rdfs:subClassOf rdf:resource = "#Information_Service"/>
10   <Input_Parameter rdf:datatype = "&Concepts;ISBN">ISBN</Input_Parameter>
11     <Output_Parameter rdf:datatype = "&Concepts;Store">Book Store</Output_Parameter>
12   <Output_Parameter rdf:datatype = "&Concepts;Price">Price</Output_Parameter>
13     <Output_Parameter rdf:datatype = "&Concepts;Currency">Result_Currency</Output_Parameter>
14   <hasPseudoTransitivity rdf:resource = "&Tasks;Currency_Converter-Task"/>
15 </owl:Class>
16 <owl:Class rdf:ID = "Currency_Converter-Task">
17   <rdfs:subClassOf rdf:resource = "#Information_Service"/>
18   <Input_Parameter rdf:datatype = "&Concepts;Price">Input_Price</Input_Parameter>
19   <Input_Parameter rdf:datatype = "&Concepts;Currency">Input_Currency</Input_Parameter>
20   < Input_Parameter rdf:datatype = "&Concepts;Currency">Output_Currency</Output_Parameter>
21   <Output_Parameter rdf:datatype = "&Concepts;Price">Output_Price</Output_Parameter>
22 </owl:Class>
23 <owl:Class rdf:ID = "BookSearch-Task">
24   <rdfs:subClassOf rdf:resource = "#Information_Service"/>
25   <Input_Parameter rdf:datatype = "&Concepts;Title">Title</Input_Parameter>
26   <Output_Parameter rdf:datatype = "&Concepts;Book">BookDetails</Output_Parameter>
27   <hasAdditivity rdf:resource = "&Tasks;BookISBN-Task"/>
28 </owl:Class>
```

TKBO=Task Knowledge Base Ontology.

performed before the ***BookPrice-Task***. The tasks dependency knowledge is semantically described for these two task instances (***BookISBN-Task*** & ***BookPrice-Task***) through <***hasTransitivity***> rule of RAAs in the semantic contexts of the task instance (***BookISBN-Task***) in Table 5 (line 5). The other task dependencies are described in the same fashion through RAAs in Table 5. In this manner, various meanings related to the task can be described in the semantic contexts of task instances and their dependency knowledge that give us an opportunity to derive a new process named ***NewBook-Process*** from these two processes. Based on this transitive characteristic of the ***hasTransitivity***, the resultant new task (***NewBook-Process)*** is performable before any suitable task to continue the possible chain. Some of possible reasoning operations are formalized as below:

$$\begin{gathered}\text{TRANSITIVITY1[BookISBN-Task(Title} \rightarrow \text{ISBN)},\\ \text{BookPrice-Task(ISBN} \rightarrow \text{Store||Price||Currency)]}\\ \text{implies [NewBook-Process(Title} \rightarrow \text{Store||Price||Currency)]}\end{gathered} \tag{1}$$

$$\begin{gathered}\text{PSEUDOTRANSITIVITY1[NewBook-Process}\\ \text{(Title} \rightarrow \text{Store||Price||Currency), Currency\_Converter-Task}\\ \text{(Currency||Price||Currency} \rightarrow \text{Price)] implies}\\ \text{[AnotherBook-Process(Title||Currency} \rightarrow \text{Store||Price)]}\end{gathered} \tag{2}$$

$$\begin{gathered}\text{PRODUCED NEW PROCESS (RESULTNODE):}\\ \text{AnotherBook-Process (Title||Currency} \rightarrow \text{Store||Price)}\end{gathered} \tag{3}$$

Table 6 Profile model of the returned composite process (AnotherBook-Process)

```
1  <tasks:AnotherBook-Task rdf:about = "AnotherBook-TaskProfile">
2  <profile:hasInput>
3     <process:Input rdf:ID = "Title">
4     <rdfs:label>Book Title</rdfs:label>
5     <process:parameterType rdf:datatype = "&Concepts;Title"/>
6     </process:Input>
7  </profile:hasInput>
8  <profile:hasInput>
9     <process:Input rdf:ID = "Currency">
10       <rdfs:label>Output_Currency</rdfs:label>
11       <process:parameterType rdf:datatype = "&Concepts;Currency"/>
12    </process:Input>
13 </profile:hasInput>
14    <profile:hasOutput>
15       <process:Output rdf:ID = "Store">
16          <rdfs:label>Book Store</rdfs:label>
17          <process:parameterType rdf:datatype = "&Concepts;Store"/>
18       </process:Output>
19    </profile:hasOutput>
20    <profile:hasOutput>
21       <process:Output rdf:ID = "Price">
22          <rdfs:label>Output_Price</rdfs:label>
23          <process:parameterType rdf:datatype = "&Concepts; Price/>
24       </process:Output>
25    </profile:hasOutput>
26 </tasks: AnotherBook-Task>
```

The profile model of the resultant composition process (Formula 3) is given in Table 6. In Table 6, two inputs (lines 2–7 and 8–13) and two outputs (lines 14–19 and 20–25) are defined with their parameter types by URLs pointing to the related concepts in the Concepts.owl (CKBO).

In Table 7, a part of process model of the resultant composite process by SCA is given. The model contains semantic contexts of the knot points in the returned chain, which are ***TRANSITIVITY1*** (Formula 1), PSEUDOTRANSITIVITY1 (Formula 2), and ***RESULTNODE*** (Formula 3). The order is described as ***TRANSITIVITY1***, ***PSEUDOTRANSITIVITY1***, and ***RESULTNODE***, which is given in lines 1–21 through <***process:ControlConstructList***>, <***list:first***>, <***list:rest***>, and so on. Three different tasks are used for the resultant composition plan: ***BookISBN-Task***, ***BookPrice-Task***, and ***Currency_Converter-Task***. The common parameter of the ***BookISBN-Task*** and ***BookPrice-Task*** is '***ISBN***'. ISBN is produced by ***BookISBN-Task*** and consumed by ***BookPrice-Task*** that is shown between lines 31 and 46 in Table 7. Common parameters of ***BookPrice-Task*** and ***Currency_Converter-Task*** are '***Input _Currency***' and '***Input_Price***'. These produced by ***BookPrice-Task*** and consumed by ***Currency_Converter-Task***, which are shown in lines 47–67 in Table 7. Finally, during the ***Execution*** and ***Monitoring*** stage, the SCA System uses grounding information (keeps the URLs of services) of those atomic processes to execute the composite process. The semantic context of the grounding information is given in lines 68–73 in Table 7. Thus, various new processes can be inferred based on the semantic contexts of task instances and their dependency knowledge in the Tasks ontology (TKBO). The system produces an OWL-S-based ontology file of requested composite process. Matching of the tasks in the resultant plan and also tasks of the client goal will be checked by the ***Process Equivalency Task***, which is discussed next.

8 Process Equivalency Task

The planner performs the ***Process Equivalency Task*** between the client goal process (G) and the newly derived composite process (P) in order to identify if the two processes have the same purpose or not. This is performed according to the algorithm depicted in Table 8. Let us assume that the input and output parameters of processes G and P are indicated as $\mathbf{G} \equiv \{\mathbf{G}_i, \mathbf{G}_o\}$ and $\mathbf{P} \equiv \{\mathbf{P}_i, \mathbf{P}_o\}$, let us say that

$\mathbf{G}_i \equiv \{$**BookTitle[Text], InputCurrency[Currency], OutputCurrency[Currency]**$\}$ and

$\mathbf{G}_o \equiv \{$**BookPrice[Price]**$\}$.

$\mathbf{P}_i \equiv \{$**Title[Text], SourceCurrency[Currency], DestinationCurrency[Currency]**$\}$ and

$\mathbf{P}_o \equiv \{$**OutputAmount[Price]**$\}$. Then clearly, $\mathbf{G}_i \equiv \mathbf{P}_i$ and $\mathbf{G}_o \equiv \mathbf{P}_o$, consequently, P is equivalent to G according to their parameter types that is given in brackets '[ ]'. In another case, if $\mathbf{G}_i \equiv \mathbf{P}_i$,

Table 7 Process model of the returned composite process (AnotherBook-Process)

```
<process:CompositeProcess rdf:about = "#AnotherBook-Process">…
  <process:ControlConstructList>
  <list:first>
  <process:Perform rdf:nodeID = "TRANSITIVITY1"/>
  </list:first>
  <list:rest>
    <process:ControlConstructList>
      <list:first>
      <process:Perform rdf:nodeID = "PSEUDOTRANSITIVITY1"/>
      </list:first>
      <list:rest>
        <process:ControlConstructList>
          <list:rest rdf:resource = "&list;#nil"/>
          <list:first>
          <process:Perform rdf:nodeID = "RESULTNODE"/>
          </list:first>
        </process:ControlConstructList>
      </list:rest>
    </process:ControlConstructList>
  </list:rest>
</process:ControlConstructList>…
```

```
<process:Process rdf:ruleID = "TRANSITIVITY1">
    <process:process rdf:resource = "../services/ BookISBN.owl#BookISBN-Process"/>
    <process:hasDataFrom>
      <process:InputBinding>
      <process:toParam rdf:resource = "../services/ BookISBN.owl#Title"/>
      …
      </process:InputBinding>
    </process:hasDataFrom>
</process:Process>
```

```
<process:Process rdf:ruleID = "PSEUDOTRANSITIVITY1">
<process:process rdf:resource = "../services/ BookPrice.owl#BookPrice-Process"/>
<process:hasDataFrom>
  <process:InputBinding>
    <process:toParam rdf:resource = "../services/BookPrice.owl#ISBN"/>
    <process:valueSource>
        <process:ValueOf>
        <process:theVar rdf:resource = "../services/ BookISBN.owl#ISBN"/>
        <process:fromProcess>
        <process:Process rdf:ruleID =  "TRANSITIVITY1"/>
        </process:fromProcess>
        </process:ValueOf>
    </process:valueSource>
  </process:InputBinding>
</process:hasDataFrom>
</process:Process>
```

```
<process:Process rdf:ruleID = "RESULTNODE">
    <process:process rdf:resource = "..services/ CurrencyConverter.owl#CurrencyConverterProcess">
    <process:hasDataFrom>
<process:InputBinding>
      <process:valueSource>
      <process:ValueOf>
      <process:fromProcess rdf:ruleID = "PSEUDOTRANSITIVITY1"/>
        <process:theVar rdf:resource = "../services/ BookPrice.owl#Price"/>
      </process:ValueOf>
    </process:valueSource>
    <process:toParam rdf:resource = "..services/ Currency_Converter.owl#Input_Price"/>
</process:InputBinding>
```

Table 7 *(Continued)*

```
59  <process:InputBinding>
60      <process:valueSource>
61      <process:ValueOf>
62          <process:fromProcess rdf:ruleID = "PSEUDOTRANSITIVITY1"/>
63          <process:theVar rdf:resource = "../services/ BookPrice.owl#Currency"/>
64      </process:ValueOf>
65    </process:valueSource>
66    <process:toParam rdf:resource = "..services/ CurrencyConverter.owl#Input_Currency"/>
67  </process:InputBinding>…

68  <grounding:WsdlGrounding rdf:about = "#AnotherBook-ProcessGrounding">
69      <grounding:hasAtomicProcessGrounding rdf:resource = "…/Currency_Converter.owl#Currency_ConverterGrounding"/>
70          <grounding:hasAtomicProcessGrounding rdf:resource = "…/BookPrice.owl#BookPriceGrounding"/>
71          <grounding:hasAtomicProcessGrounding rdf:resource = "…/BookISBN.owl#BookISBNGrounding"/>
72          <service:supportedBy rdf:resource = "#AnotherBook-ProcessService"/>
73  </grounding:WsdlGrounding>
```

Table 8 Process Equivalency Task

1	When Gi $\equiv$ Pi then
2	When Go $\equiv$ Po then G $\equiv$ P
3	When Go $\subset$ Po then G $\equiv$ P
4	When Go $\supset$ Po then G $\neg\equiv$ P
5	otherwise G $\neg\equiv$ P
6	When Gi $\supseteq$ Pi then
7	When Go $\equiv$ Po then G $\equiv$ P
8	When Go $\subset$ Po then G $\equiv$ P
9	When Go $\supseteq$ Po then G $\neg\equiv$ P
10	otherwise G $\neg\equiv$ P
11	When Gi $\subset$ Pi then G $\neg\equiv$ P

however, $\mathbf{G}_o \equiv \{$**BookPrice[Price]**$\}$ and **Po** $\equiv \{$**OutputAmount[Price], DestinationCurrency[Currency]**$\}$, that is, $\mathbf{G}_o \subset \mathbf{P}_o$, then it can be stated that P subsumes process G. Producing extra outputs is acceptable, therefore it can be concluded that P is equivalent to G. In the latter case, the SCA System checks whether any unnecessary I/O is generated, and if so, stipulates where it can be ignored in the above example.

9 Related works

In recent studies, the composition problem is solved by using a planner. A planner uses a list of participating candidate Web services (*contains atomic or composite processes*) and a complex goal (*stated by the user in the form of a task description*) in generating a composition plan. The most popular AI planning techniques are the Hierarchical Task Network (HTN) planner using Situation Calculus (SC) as discussed by (Sirin, E. & Parsia, B., 2004; Sirin *et al*., 2004). Event Calculus (EC)-based composition as discussed by Aydin *et al*. (2006, 2008); Planning Domain Definition Language (PDDL)-based composition as introduced by Yang and Qin (2010); and Compositional/Process Algebra (PA, Bergstra *et al*., 2001)-based composition as given by Hashemian & Mavaddat, 2006; Forgy, 1991).

HTN planner (Sirin *et al*., 2004) uses a task decomposition approach, called the fragmentation process that divides complex tasks into smaller subtasks until it reaches atomic tasks that can be executed directly. The advantage of the HTN method is that it decreases the complexity of planning phase that may require many steps, however, the mechanism of dividing high-level tasks is in itself difficult to execute accurately.

The basic ontology of EC is very similar to that of SC. Aydin *et al.* (2006, 2008) contributed a valuable approach to the problem of composition of Web services by using the EC method. This work presented a method of translating the OWL-S ontology to EC events and actions. The main idea is to get a suitable composition plan with complex actions by using EC as an alternative approach to building agent technology, based on the notions of generic procedures and customizing user constraints.

The PDDL; (McDermott, 1998) is widely documented as a standardized input for high-tech planners and used for composition of Web services. Consequently, the mapping process from DAML-S or OWL-S representations to PDDL is very user friendly and simple to operate because the ontology languages have been strongly influenced by PDDL and vice versa. Yang and Qin (2010) proposed that the OWL-S description of SWSs can be translated into a description written in PDDL. Also they developed several key procedures of the translating algorithm for OWLS2PDDL and vice versa.

Hashemian and Mavaddat (2006) represented the behavior and operations of a Web service using a simple PA, called '*Composition Algebra*'. PA operators are used to find a solution for automatic composition from stateless components, which had a simple two-step workflow: receiving inputs and then returning the corresponding outputs. They provided a forward chaining

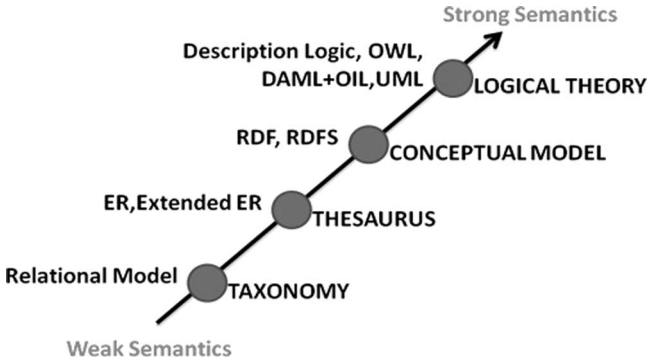

Figure 2 Ontology spectrum: weak to strong semantics (after Obrst, 2003).

algorithm to find possible components for composition in order to achieve a requested behavior. However, the proposed approach is not modeled on Semantic Web, thus lacks semantics.

Most researchers concentrated either on semi-automated or fully automated techniques for Web services composition, drawing inspiration particularly from AI planning and Process Calculi. However, these approaches are lexical in nature and can only return service composition upon user's necessity description, which lacks flexibility in meeting later change. Two important contributions of this present study are that the SCA takes advantage of its own TKBO, and RAAs based inferencing. Using TKBO, Web services can be defined as instances of classes by using OWL semantic tags that represent Web services capabilities based on their OWL-S descriptions. A domain-specific TKBO contains commonly used task instances for a specific domain (e.g. Online Book Selling/Buying, Traveling, or Car Selling/Buying, etc.), I/O/P/E concepts of the described task instances, and semantically described task dependencies through RAAs by using OWL tags (e.g.Table 5 contains four task instances of the Online Book Store Domain). Additionally, SCA keeps the semantic contexts of task dependencies belonging to the newly constituted composite task in the corresponding domain Tasks ontology (TKBO) for possible reuse. For example, Tables 6 and 7 display Profile and Process models of the returned composite process (*AnotherBook-Process*). The technique saves time and effort whenever same composite task is requested or may be used to form more complex composite tasks later.

Additionally, in order to select suitable tasks for the required composition, the SCA searches over lesser number of the related task instances that are in the interest domain TKBO instead of searching over a large number of candidate OWL-S (POKB) processes at first. After specifying the suitable task instances for newly constituted appropriate solution plan by Planner, the SCA selects same structured OWL-S atomic processes (*a task instance is having same I/O/P/E with a candidate OWL-S atomic process*) from candidate set by looking only the included task instances in the plan. After execution of the resultant plan, the semantic contexts of the newly produced composite task will have been embedded into the TKBO if the newly produced composite task was not produced and recorded before. Therefore, the SCA does not require to search and parse the large set of candidate OWL-S processes, which makes the performance better compared with other solutions in this case. Although many proposed solutions employ Semantic Web technologies, however, they do not take advantage of such a common TKBO.

Moreover, the SCA has a different and extended matching mechanism since the matching task takes into consideration similarity and defines relations such as ***is_a, synonym***, or ***equivalentClass*** between I/O concepts of processes. Additionally, the process matching task considers both properties that are ***rdf:ID*** *(matching of input- output)* concepts and ***process:parameterType*** properties (see details in Section 4). Consequently, relevant combinative processes will not be missed in finding a composition plan.

As mentioned above, the SCA System draws advantages from its OWL ontology knowledge bases due to their expressive nature and semantic richness. SCA uses the RAAs to further

elaborate on such rich knowledge bases. In fact, AAs being created for the relational models, this may bring to one's mind the question: How appropriate is to employ RAAs/AAs in connection with OWL ontologies. In order to address this issue, and relate AAs to OWL, let us study the ontology spectrum. Obrst (2003) developed an ontology spectrum as a framework for illustrating various information models relative to others in terms of their semantic richness. An abridged version of the Obrst ontology spectrum as depicted in Figure 2 lines up some existing mainline semantic languages or models by plotting them on a span from weak to strong semantics. Obrst says that 'Weak semantics indicate the expression of very simple meaning while strong semantics indicate the expression of arbitrarily complex meaning. As one progresses up the spectrum, the domain semantics are based on logic, allowing the machine to make valid inferences and execute sound semantic constraints' (Obrst, 2003).

All models of the given spectrum have an association with each other that correspond to semantic expressiveness and syntax; for instance: the Description Logics (DL) syntax can be used for a description of OWL ontology or knowledge base. A concept from DL is referred to as a class in OWL and a role from DL is referred to as a property in OWL. Furthermore, the Relational Model (RM) has a strong mathematical foundation. This makes it closer to the DL layer, which is the basis for many ontology languages, including OWL. In addition, tables in the RM are equivalent to ontology classes. The main advantage with respect to ER is that the relational model includes domains for the attributes (Fagin, 1977; Abiteboul *et al*., 1995; Rao & Su, 2005; Motik *et al*., 2007). Higher level semantic models on the spectrum provide more expressiveness, semantic richness and complexity than those in the lower levels. On the other hand, while climbing from the bottom to the top, there will be no semantic loss. To the contrary, more powerful descriptions of functions, data and properties will be availed. The *AAs* are well known and widely used in inference rules in the area of the Relational models and Databases. Consequently, it is just to say that, AAs may as well be a suitable approach to derive tasks dependencies of a composition as used in Sections 6 and 7 in relation to SWSs.

10 Conclusions

Lack of semantic annotation parts, increasing number of Web services on the Web, and syntactic-based search operations for current Web services makes discovery and composition of appropriate Web services challenging. This article presented *an inference-based SCA* approach to perform automatic process composition of SWSs. The proposed SCA System has five main parts, which are as follows: translator, planner, inference engine, execution engine, and monitoring agent. The system also has three ontology knowledge bases; Tasks-TKBO, Concepts (or Domains-CKBO) and Processes-PKBO, and one RKB that contains the RAAs.

SCA starts with a scheme to parse specifications of tasks instances in TKBO, interested OWL-S-based atomic processes and goal task to I/O/P/E form, then uses RAAs in building a planner-inference engine cycle. A semantically enriched algebraic rule-based planner was developed for use in composing atomic processes of candidate SWSs. Examples are given displaying practical uses of functional parts of the SCA System approach. A case study for generating composition plans were performed displaying the outcome using different OWL-S atomic processes selected from the widely used SWS-TC 1.1 test collection.

Acknowledgement

This work was supported in full by the Eastern Mediterranean University Scientific Research Project BAP-A-07-20 *Provision of SWSs through an Intelligent Semantic Web Service Finder*.

References

Abiteboul, S., Hull, R. & Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley, 0-201-53771-0.

Armstrong, W. W. 1974. *Dependency Structures of Data Base Relationships* Information Processing **74**. North-Holland Pub. Co, 580–583.

Aydin, O., Cicekli, N. K. & Cicekli, I. 2006. Towards automated Web Service composition with the abductive event calculus. In *Proceedings of Applications of Logic Programming in the Semantic Web and Semantic Web Services (ALPSWS 2006)*, 103–104, Seattle, USA.

Aydin, O., Cicekli, N. K. & Cicekli, I. 2008. *Automated Web Services Composition with the Event Calculus*. Engineering Societies in the Agents World VIII (ESAW 2007), Lecture Notes in Computer Science, Springer Berlin/Heidelberg Press, 0302-9743, 142–157.

Berners-Lee, T., Hendler, J. & Lassila, O. 2001. *The Semantic Web*. Scientific American.

Celik, D. & Elci, A. 2008. Provision of Semantic Web Services through an intelligent Semantic Web Service finder. *Multiagent and Grid Systems—An International Journal* **4**(3), 1574–1702. IOS Press, 315–334.

Fagin, R. 1977. Functional dependencies in a relational data base and propositional logic. *IBM Journal of Research and Development* **21**(6), 543–544.

Forgy, C. L. 1991. *Rete: a fast algorithm for the many pattern/many object pattern match problem*, Expert systems: a software methodology for modern applications, IEEE Computer Society Press, Los Alamitos, CA, 324–341.

Hashemian, S. V. & Mavaddat, F. 2006. *Composition algebra: process composition using algebraic rules*. Third International Workshop on Formal Aspects of Component Software (FACS'06), Prague, Czech Republic.

Kang, D., Lee, S., Kim, K. & Lee, J. Y. 2009. An OWL-based semantic business process monitoring framework. *Journal of Expert Systems with Applications* **36**(4), 0957–4174.

McDermott, D. 1998. *The Planning Domain Definition Language Manual*. Yale Computer Science Report **1165** (CVC Report 980003).

McIlraith, S. A., Son, T. C. & Zeng, H. 2001. Semantic Web Services. *IEEE Intelligent Systems* **16**, 46–53.

Motik, B., Horrocks, I. & Sattler, U. 2007. Bridging the gap between OWL and relational databases. In *Proceedings of the 16th international conference on World Wide Web*, 807–816.

Obrst, L. 2003. Ontologies for semantically interoperable systems. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, 366–369.

OWL. 2004. *OWL Web Ontology Language Overview: W3C Recommendation*. Retrieved October 1, 2009. Available at http://www.w3.org/tr/owl-features/

OWL-S. 2004. *Semantic Markup for Web Services: W3C Recommendation*. Retrieved October 1, 2009. Available at http://www.w3.org/submission/owl-s/

Rao, J. & Su, X. 2005. *A Survey of Automated Web Service Composition Methods*. Lecture Notes in Computer Science, **3387**, 0302-9743 (Print) 1611-3349 (Online), 43–54.

Sirin, E. & Parsia, B. 2004. Planning for Semantic Web Services. In S*emantic Web Services Workshop at 3rd International Semantic Web Conference*.

Sirin, E., Parsia, B., Wu, D., Hendler, J. & Nau, D. 2004. HTN planning for Web service composition using SHOP2. *Journal of Web Semantics* **1**(4), 377–396.

Yang, B. & Qin, Z. 2010. Composing semantic Web services with PDDL. *Journal of Information Technology* **9**(1), 48–54.