Analysis of Enterprise Architecture Models with Description Logics Reasoning and SPARQL

Gonçalo Antunes, Instituto de Engenharia de Sistemas e Computadores – Investigação e Desenvolvimento em Lisboa, Portugal, goncalo.antunes@inesc-id.pt
Artur Caetano, Instituto Superior Técnico, Universidade de Lisboa, Portugal, artur.caetano@tecnico.ulisboa.pt
José Borbinha, Instituto Superior Técnico, Universidade de Lisboa, Portugal, jlb@tecnico.ulisboa.pt

Abstract

Enterprise architecture (EA) model analysis can be defined as the application of property assessment criteria to EA models. Ontologies can be used to represent conceptual models, providing means for their integration, at the semantic and syntactic level, and allowing the application of computational inference to derive logical conclusions from the facts present in the models. As the commonly used EA modelling languages are conceptual, advantage can be taken of representing such conceptual models using ontologies. This paper explores and demonstrates the use of ontologies and associated techniques in the analysis of enterprise architecture models. Two techniques are used to this end: computational inference and the use of SPARQL. The aim is to demonstrate the possibilities brought by the use of these techniques in EA model analysis.

Keywords: Enterprise architecture, model analysis, ontologies, reasoning, SPARQL

1. INTRODUCTION

Enterprise architecture (EA) model analysis has a focus on the application of techniques that process the artefacts, properties and dependencies of a model, generating information that can be used to assess, transform or redesign organizational systems [Bucher et al. 2006], [Johnson, Lagerstrom et al. 2007]. Model analysis can be seen as the “application of property assessment to models” with the goal of observing the system’s functional and non-functional qualities [Bertolino et al. 2013].

Ontologies enable representing the aspects of a conceptual model, which is composed of a conceptual schema and an information base [Olivé 2007]. Ontologies also provide the mechanisms to integrate different schemas through the definition of rules that relate the concepts at structural or semantic level. Moreover, with logic-based ontologies, inference can be used to derive logical conclusions from the ontological facts and support model analysis.

Enterprise architecture models expressed in the actual common EA languages can be expressed using ontologies. Hence, advantage can be taken of the features of ontologies, namely the integration and the computational inference mechanisms. Moreover, different techniques to apply to this end are widely available as part of the semantic web standards and frameworks. OWL [W3C 2012] can be used to represent
ontologies, and the particular sublanguage of OWL-DL is in fact an implementation of description logics, which can be reasoned upon using one of the several reasoners available. Available semantic web frameworks such as the Apache Jena allow the storing and exposing of ontologies in a high-performance triple-store, with APIs for interfacing with other systems and reasoners, and SPARQL [W3C 2013] query facilities for accessing and manipulating the data contained in the ontologies.

This work explores the use of ontologies in the analysis of enterprise architecture models. In particular, two techniques are targeted to this end: the use of computational inference to reason upon the models and deduce implicit knowledge, and the use of SPARQL to retrieve and manipulate the data stored in the models. The use of these techniques is made on top of a federated model for the integration and analysis of EA models previously published in [removed for blind review]. In order to demonstrate the application of the techniques, a fictional scenario modelled using the ArchiMate language will be used.

This paper is organized as follows. Section 2 briefly describes related work in EA analysis, while Section 3 provides an overview in Ontologies, including description logics and OWL. Then, in Section 4, the referred ontology related techniques are demonstrated in performing different types of EA model analysis. Finally, the paper concludes in Section 5.

2. ENTERPRISE ARCHITECTURE ANALYSIS

Enterprise architecture analysis assists the continuous enterprise architecture program by providing information to support the planning, improvement, and management processes [Hamalainen 2008]. It also supports the governance, optimization, re-engineering and decision-making processes of an organization [Hamalainen 2008], [Lankhorst 2005], [Johnson and Ekstedt 2007]. According to [Niemann 2005], EA analysis can be classified according to the following categories:

- Dependency analysis relates enterprise architecture artefacts to derive direct and indirect dependencies between them;
- Coverage analysis detects redundancies and gaps existing in an EA description, such as missing artefacts or missing relationships between them;
- Interface analysis assesses how the interfaces of artefacts relate usually with the goal of determining the degree of coupling and cohesion;
- Heterogeneity analysis is used for determining the identifies elements that need re-factorizing as means to homogenize the overall architecture description;
- Complexity analysis is typically used for determining the number of elements existing in an EA description and the number of relationships they have;
Compliance analysis determines if the artefacts or the overall architecture description meet policies, rules and requirements;

Cost analysis calculates the costs of an artefact (e.g. creation, maintenance) or costs pertaining to the architecture description;

Benefit analysis determines the contribution of an artefact to the overall goals of the organization as described in architecture.

We can find in the literature specific proposed approaches for this purpose. In [Johnson et al. 2004], the addition of non-functional attributes to enterprise architecture models through architecture theory diagrams (ATD) is proposed. These diagrams interrelate attributes through composition, correlation, and casual relationships. The ATD is then populated with measures that support the calculation of the model’s non-functional attributes. Extended influence diagrams (EID) [Johnson, Lagerstrom et al. 2007] and probabilistic relational models (PRM) [Lagerstrom et al. 2010] are other analysis techniques. EID are used to model goals and decision alternatives, thus providing support for decision making. EID use probabilistic inference to support the representation of uncertainty when computing the values of attributes. PRM extend entity relationship models, and support model analysis under uncertainty.

In [Davoudi and Sheikhvand 2012], the analytical hierarchical process (AHP) is proposed to prioritize and select architectural scenarios according to the non-functional requirements under analysis. In the work reported in [Franke, Flores and Johnson 2009], fault tree analysis (FTA), an extended Bayesian network, is used to analyze dependencies related to reliability and reusability qualities. The use model mapping and transformation to analyze the compliance of process models against a set of actor coordination patterns is reported in [Caetano, Assis and Tribolet 2012]. In [de Boer et al. 2005], an XML schema is used to encode an enterprise architecture meta-model that specifies the structure and dynamics of enterprise architecture models. This approach analyses the structure of a model in terms of its cardinality, class specialization, and concept relationships. The dynamic analysis uses scenarios that encode state-based actions with XML and RML rules to simulate the behavior of the architecture.

3. ONTOLOGIES

An ontology is defined as a “formal, explicit specification of a shared conceptualization” [Studer, Benjamins and Fensel 1998]. According to [Guarino, Oberle and Staab 2009] and [Genesereth and Nilsson 1987], “conceptualization” refers to an “abstract, simplified view of the world”, containing “the objects, concepts, and other entities that are assumed to exist in some area of interest and the relationships that hold among them”. In [Studer, Benjamins and Fensel 1998], “explicit” is related to the definition of the “type of concepts used, and the constraints on their use”, while “formal” to the fact that the conceptualization “should be machine readable”. Finally, “shared” means that the ontology “captures consensual knowledge”.
Description logics (DL) are “a family of logic-based knowledge representation languages suitable for the representation of ontologies”, which can be seen as “a decidable fragment of first-order logic” [Vaculin 2009]. DL describe domains in terms of concepts, roles, and individuals. Roles and concepts are related using logical statements named axioms. Different varieties of DL exist with differing degrees of expressiveness. According to [Areces 2000], DL supports five different types of reasoning: subsumption, instance checking, relation checking, concept consistency, and knowledge base consistency. Subsumption organizes concepts in a hierarchy and finds the most specific super-class for each class. Instance checking verifies if a given individual is an instance of a concept. Relation checking verifies if and how two individuals relate to each other. Concept consistency verifies if there are no contradictions between the definitions or the chain of definitions of a concept. Finally, knowledge base consistency determines whether the information contained in the knowledge base contains any contradictions.

The Web Ontology Language (OWL) is described in [W3C 2012] as a “semantic web language designed to represent rich and complex knowledge about things, groups of things, and relations between things”. OWL can be used in a varying level of expressiveness, with a particular variant known as OWL-DL, being an implementation of DL which retains “computational completeness” [W3C 2004]. The terminology used in OWL is different from that used in DL: a DL concept corresponds to an OWL class, a DL role corresponds to an OWL property, and a DL individual corresponds to an OWL object. An OWL-specified ontology is interpreted as a set of “objects” and a set of “properties” which relate objects with each other. Ontologies expressed in OWL consist of axioms that constrain the classes and their relationships. Axioms allow making explicit information that otherwise is implicit through the use of logical inference. Properties are used to state relations between individuals or between an individual and a data value. There are two main categories of properties: Object properties, which “link individuals to individuals”, and Data properties, which “link individuals to data type values”. Ontologies specified in OWL-DL follow the “open world assumption”. This means that if a meta-model represented as an ontology does not state clearly axioms that enable to state a fact as being true, then it is considered to be undefined.

OWL ontologies can be represented using RDF (Resource Description Framework) [W3C 2014] triples. RDF is a W3C standard compliant format in which any ontology can be represented. The data represented using RDF consists of triples, each comprising of subject, predicate and object. A set of triples represents a graph database that can be queried using SPARQL [W3C 2013]. SPARQL is a query language which allows retrieval and manipulation of data stored in RDF. It can only retrieve data stored in the model and, contrary to the DL language, there is no inference in the language itself. The information retrieved from the model can be provided in three different ways: raw values in a table format (using SELECT query), RDF triples which are a subset of the queried model (using CONSTRUCT query), or a Boolean value. The selection of the kind of query mode depends on the particular needs. Additionally, SPARQL allows the use of aggregate
operations, which can be used to apply expressions over groups of queried data. Count, sum, min, max, average, group concatenation and sampling are the supported operations.

4. ANALYSIS WITH DESCRIPTION LOGICS REASONING AND SPARQL

Based on the federated arrangement proposed in [Antunes et al. 2014], the different types of EA model analysis will be demonstrated and described below using an example EA description and a determined EA meta-model, which will be extended when needed for demonstration purposes. The demonstration will be based on the use of the ArchiMate modelling language and will use a fictitious scenario comprising several elements belonging to the different abstraction layers depicted in ArchiMate. Figure 1 depicts a layered view of the scenario ArchiMate model.

As such, the ArchiMate meta-model, along with the Motivation and Implementation and Migration extensions were specified in OWL-DL. The method employed for creating an ArchiMate representation in OWL-DL involved three steps: (i) transform the ArchiMate meta-model; (ii) adding axioms and cardinalities; and (iii) transforming the ArchiMate models. In step (i), an analysis of ArchiMate's meta-model was performed concept-by-concept, including the relations with other concepts and the constraints existing in those relations. The result was the mapping of concepts into OWL Classes and the mapping of relations into OWL Object Properties.

In step (ii), restrictions were added to the properties, such as Inverse Object Properties and Super Object Properties axioms. Axioms were added to ensure the compliance against the ArchiMate specification, including cardinality. Finally, Step (iii) is scenario dependent and involves creating objects of the classes existing in the ArchiMate ontology created in the two previous steps, which correspond to the elements modelled in an ArchiMate model.
4.1. Dependency Analysis

Taking advantage of the possibility of introducing semantics on the models brought by the use of a federated approach, two Super Object Property chains were created for modelling dependencies between different elements. The `dependsDown` object property is thus a Super Object Property of the `uses`, `realizedBy`, `aggregates`, `composes`, and `assignedTo` Object Properties that resulted from the conversion of the ArchiMate relations, while the `dependsUp` object property fills the same purpose for the counterpart Inverse Object Properties: `aggregatedBy`, `composedOf`, `usedBy`, `realizes`, and `assignedFrom`. Moreover, these properties are transitive, which makes possible the creation of a graph of dependencies.

The definition of the dependency object properties allows retrieving answers to the following questions: what are the technological entities supporting the Business Service 1? Such a question can be formulated in DL as follows:

```
Thing and hasLayer some TechnologyLayer and dependsUp value Business_Service_1
```
Figure 2 depicts the answer to the query using the HermiT reasoner in Protégé. As can be seen in the figure, different elements pertaining to the technology layer were identified as being dependencies of the Business Service 1.

![Figure 2 - Dependency analysis using DL reasoning](image)

Figure 3 depicts the explanation given by the reasoner as why a certain node is a dependency of that service, where it is possible to see the realizes object property and the usedBy object property as being sub object properties of the dependsUp object property and the fact that this property is transitive.

![Figure 3 - Reasoner explanation for dependency.](image)

The use of SPARQL on top of the inferred model further allows the use of aggregates. In this case, the number of technological dependencies of the aforementioned service can be computed using the below query, which returns the result “6”.

![Query to compute number of dependencies](image)
4.2. Coverage Analysis

The analysis of the coverage of an architecture typically tries to find gaps in the architecture, i.e. elements or relationships that are wrongly missing from the architecture description. Although the use of reasoning on top of DL allows verifying the existence of individuals of a certain class or the existence of a relationship between two individuals, the open world assumption associated with OWL does not allow us to test for the non-existence of a relationship between two individuals, unless it is explicitly specified.

However, SPARQL can be used to this end. The SPARQL query below exemplifies the verification of the existence or not of application support provided to processes. It uses the particular query form ASK, which returns a Boolean depending on the verification of the triple patterns specified in the query. For instance, the execution of the query as exemplified below, i.e., for Business Process 7, returns precisely a Boolean value of “true”, while if executed for Business Process 8, it will return the value “false”.

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX uo: <http://sysresearch.org/ontologies/UO.owl#>

ASK {
  FILTER EXISTS {?subject uo:dependsUp ?object} .
}
```

4.3. Interface Analysis

The use of a DL reasoner can be a way of revealing the elements that interface, explicitly or implicitly, with a given element. However, this can also be performed with the use of SPARQL queries. Queries on top of
a non-inferred ontology reveal explicit relationships, and queries on top of the inferred model make explicit otherwise implicit relationships. The query below can be used for determining the interfacing elements of Business Process 6, with Figure 4 revealing the results when executed on top of the non-inferred model.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX uo: <http://sysresearch.org/ontologies/UO.owl#>

SELECT ?label1 ?predicate ?label2
WHERE {
    {?subject rdfs:label "Business_Process_6"^^xsd:string} UNION
    {?object rdfs:label "Business_Process_6"^^xsd:string} .
}
```

![Figure 4 - Interface analysis SPARQL query results](image)

### 4.4. Heterogeneity Analysis

As previously referred, the analysis of the heterogeneity of the architecture can take advantage of the subsumption and instance checking reasoning tasks that can be performed on top of a DL- specified ontology. In practice, the reasoner can be used to show the hierarchical arrangement of the classes existing on the architecture ontology, and it can be further enquire to show all the individuals, given a determined class.

Nonetheless, the use of SPARQL can provide interesting insights in a tabular format. The SPARQL query below can be used to return all the individuals per class of the scenario being addressed. Figure 5 depicts an excerpt of the results to the query.
4.5. Complexity Analysis

One possible measure of complexity in enterprise architecture can be given by the number of elements and relationships present in the architecture description. With the use of SPARQL, it is possible to perform simple calculations through the use of aggregates. The query below can thus be used for counting the number of individuals pertaining to different classes in the scenario ontology.
The number of relationships in which each individual of a particular class, in this case Application Component, participates in can be obtained with the query below.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX uo: <http://sysresearch.org/ontologies/UO.owl#>

SELECT ?instancel ?predicate (COUNT(?object) as ?n_relationships)
WHERE {
} GROUP BY ?instancel ?predicate ORDER BY DESC(COUNT(*))
```

![Query Results Table](image)

**Figure 6 - Complexity analysis SPARQL query results excerpt.**
Moreover, the query can be modified to target a specific type of relationship, in this case, the dependencies defined in Section 4.1, as can be seen below.

```sql
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX uo: <http://sysresearch.org/ontologies/UO.owl#>

SELECT ?instance1 (COUNT(?object) as ?n_dependencies) 
WHERE {
  {?predicate rdf:subPropertyOf uo:dependsUp} UNION {?predicate rdf:subPropertyOf uo:dependsDown}.
}
GROUP BY ?instance1 ORDER BY DESC(COUNT(*))
```

### 4.6. Compliance Analysis

In order to demonstrate this type of analysis, let us imagine that in this fictional scenario there is one requirement that concerns the availability of the infrastructure service delivered by Node 1. The requirement states that the availability of the Infrastructure Service 1 should never fall below 90%. Such indicator could be modelled as an attribute of the service. However, the ArchiMate concepts have no pre-defined attributes.

As such, using the aforementioned federated arrangement, a specialization of the concept of Infrastructure Service can be integrated with the main model, augmenting the class `InfrastructureService` with the data property `hasAvailability`. Then, through the use of automatic service monitoring tools, the value of such data property can then be regularly updated with real data concerning the availability indicator.

As the Requirement class already exists in the ontology originating from ArchiMate, the aforementioned requirement would be created as the object `AvailabilityRequirement` with the following type:

```
Requirement and (realizedBy only InfrastructureService and (has Availability only int[>= 90]))
```

The requirement would be associated with the Infrastructure Service 1 through an object property assertion `realizedBy Infrastructure_Service_1`. Assuming that the service monitoring kept feeding the data property value with the real availability of the service and with the reasoner continuously processing the ontology, it is possible to continuously monitor the compliance of the architecture with the requirement.

Therefore, if the value drops below 90, the reasoner will report an inconsistency in the ontology therefore indicating that compliance with that particular requirement is not met. Figure 7 depicts an inconsistency being detected by the HermiT reasoner and explained in Protégé.
4.7. Cost Analysis

The demonstration of this type of analysis also involves augmenting the scenario’s architecture description with cost related information. In this case, this can be achieved with the extension of the architecture description with cost information associated with the infrastructure nodes modeled in the scenario through a data property cost with a data range of the type double. Table 1 depicts the associated cost information associated with each node.

<table>
<thead>
<tr>
<th>Element</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
<th>Node 4</th>
<th>Node 5</th>
<th>Node 6</th>
<th>Node 8</th>
<th>Node 9</th>
<th>Node 10</th>
<th>Node 11</th>
<th>Node 12</th>
<th>Node 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>10.5</td>
<td>3.2</td>
<td>8.2</td>
<td>15.0</td>
<td>13.2</td>
<td>2.2</td>
<td>7.1</td>
<td>10.1</td>
<td>5.0</td>
<td>27.0</td>
<td>4.5</td>
<td>10.3</td>
</tr>
</tbody>
</table>

Table 1 – Cost information associated with nodes.

The addition of this information coupled with dependency information on top of the inferred ontology makes possible the execution of relevant analysis. For instance, the question “what is the cost associated with the business processes of the scenario?” can be answered with the use of the following SPARQL query, which makes use of the aggregate SUM. The results of the query can be observed in Figure 8.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX uo: <http://sysresearch.org/ontology/uo.owl#>
PREFIX demo_scenario: <http://sysresearch.org/ontology/scenarios/demo_scenario.owl#>

SELECT ?labell (SUM(?cost) as ?COST)
WHERE {
} GROUP BY ?labell ORDER BY ?COST
```
Figure 8 - Cost analysis SPARQL query results

4.8. Benefit Analysis

Benefit analysis is another example that requires some extension to the original scenario architecture description. In this case, the extension involves the addition of business process goal information to the existing scenario. Figure 9 depicts a graph with the new added goals related to the business processes of the scenario.

Moreover, the ArchiMate classes Goal and Business Process will be impacted, being extended with data properties that can be used for defining goal metrics. For this example, the data property defined is income, which will have the range of double, and will be used for defining income metrics for business processes. In particular, the individual Goal 6 will have this property with the value “30”.

For the sake of this demonstration, let us imagine that the business processes in Table 2 have the following income values.
Figure 9 - Graph of goals associated with business processes.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Income</td>
<td>12.0</td>
<td>7.5</td>
<td>3.1</td>
<td>16.3</td>
</tr>
</tbody>
</table>

Table 2 - Income information associated with business processes.

Using the inferred model, the income generated by business processes realizing Goal 6 can be verified with the SPARQL query below that takes advantage of dependency information on top of the inferred model.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX uo: <http://sysresearch.org/ontologies/UO.owl#>
PREFIX demo_scenario: <http://sysresearch.org/ontologies/scenarios/demo_scenario.owl#>

SELECT ?label ?income
WHERE {
}
```

In order to determine if Goal 6 is being met, a SPARQL query in the ASK form can be elaborated as displayed below. Note the use of the HAVING expression that filters the resulting triples to be evaluated in the ASK query. If the sum of the income generated by processes is lower than the income defined as the goal, no
triples are returned and the ASK query will be evaluated as false. In this case, the income generated by the processes is higher than the set income goal and hence the query will be evaluated as true.

```sparql
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX uo: <http://sysresearch.org/ontologies/UO.owl#>
PREFIX demo_scenario: <http://sysresearch.org/ontologies/scenarios/demo_scenario.owl#>

ASK
WHERE {
  demo_scenario:Goal_6 demo_scenario:income ?goalinc.
}
GROUP BY ?INCOME ?goalinc
HAVING (SUM(?income) >= ?goalinc)
```

Finally, in order to determine the contribution of each process to the goal as a percentage, we can use a SPARQL query in the SELECT form, as shown below. Note the use of the BIND expression that allows binding the resulting value of the calculation of the percentage of the contribution to a variable that can be then displayed in the results. Figure 10 depicts the resulting percentages for the example.

```sparql
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX uo: <http://sysresearch.org/ontologies/UO.owl#>
PREFIX demo_scenario: <http://sysresearch.org/ontologies/scenarios/demo_scenario.owl#>

SELECT ?process ?percentage
WHERE {
  demo_scenario:Goal_6 demo_scenario:income ?goalinc.
  bind((?income * 100)/?goalinc as ?percentage)
}
```

Figure 10 - Benefit analysis SPARQL query results.
5. CONCLUSION

This paper demonstrated the application of ontologies and associated techniques in the analysis of EA models, namely the use of computational inference to reason upon the models and deduce implicit knowledge, and the use of SPARQL to retrieve and manipulate the data stored in the models. The demonstration involved the use of practical examples of the application of the mentioned techniques for performing different types of analysis, in the context of a fictional scenario.

We believe that through this work some insights were given at the possibilities brought by the use of the techniques in scope. By representing the models using logic-based ontologies, it is possible to perform analysis just by using the syntactic and semantic information contents of the models and reason upon it. The use of SPARQL widens the possibilities, since besides allowing for the retrieval of information, it allows performing simple calculations on the retrieved data. Moreover, it can be used in conjunction with computational inference, as the queries can be applied on top of an inferred model.

6. ACKNOWLEDGEMENT

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with references UID/CEC/50021/2013 and EXCL/EEI-ESS/0257/2012 (DataStorm).

REFERENCES


