An Application of Semantic Techniques to the Analysis of Enterprise Architecture Models

Gonçalo Antunes
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
goncalo.antunes@tecnico.ulisboa.pt

Artur Caetano
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
artur.caetano@tecnico.ulisboa.pt

José Borbinha
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
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, allowing the application of computational inference to derive logical conclusions from the facts present in the models. As the actual common EA modelling languages are conceptual, advantage can be taken of representing such conceptual models using ontologies. Several techniques for this purpose are widely available as part of the semantic web standards and frameworks. This paper explores the use of the aforementioned techniques in the analysis of enterprise architecture models. Namely, 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.

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 [6], [16]. 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 [6].

Ontologies enable representing the aspects of a conceptual model, namely its conceptual schema and information base [21]. 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 [24] can be used to represent ontologies and can be reasoned upon using one of the several reasoners available. Available semantic web frameworks such as the Apache Jena1 allow the storing and exposing of ontologies in a high-performance triple-store, with APIs for interfacing with other systems and reasoners, and SPARQL [25] 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 [1]. 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. In Section 4, the federated model for EA model analysis is briefly described. Then, in Section 5, the referred ontology related techniques are demonstrated in performing different types of EA model analysis. Finally, the paper concludes in Section 6.

2. Enterprise Architecture Analysis

Enterprise architecture analysis assists the continuous enterprise architecture program by providing information to support the planning,
improvement, and management processes [13]. It also supports the governance, optimization, re-engineering and decision-making processes of an organization [13] [19] [15]. According to [20], 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-factoring 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 [14], 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) [16] and probabilistic relational models (PRM) [17] 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 [8], 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 [9], 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 [7]. In [4], 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” [22], being Studer's definition, which is based on Gruber's [11] and Borst's [5], likely the most commonly used. According to [12] and [10], conceptualisation 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”. [22] relate explicit 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” [23]. 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 [2], 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 [24] 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. 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) [26] 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 [25]. 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. A Federated Model for Analysis

In order to enable the analysis of EA models using ontologies, the federated model described in [1] is used. It comprises the concept of Upper Ontology (UO), Domain-Specific Ontology (DSO), and Map Ontology (MO). A UO is a core, high-level, EA meta-model.

A DSO is a domain-specific meta-model that provides adequate information, extending the information present in the UO. Finally, the MO is an ontology mapping the concepts of the DSO to the concepts of the UO. A variable number of DSOs can map to a UO through the use of a variable number of MOs. Such a federated arrangement allows for the enrichment of the overall EA description adding information that can enable analysis, which can then make use of different techniques for retrieving information from the models, namely reasoning and querying techniques.

Figure 1. Federated model for analysis

Figure 1 depicts a conceptualization of the aforementioned model on the left side, with a federation example on the right side. It also shows on the right side, in yellow arrows, the different reasoning configurations that can take place on top of this model, which can be at the level of a sole ontology (UO and DSO reasoning) or cross-ontology (UO-DSO and DSO-DSO reasoning).

Table 1. Reasoning support for EA model analysis

<table>
<thead>
<tr>
<th>Reasoning</th>
<th>Subsumption</th>
<th>Instance checking</th>
<th>Relation checking</th>
<th>Concept consistency</th>
<th>KB consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependency</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coverage</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Heterogeneity</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complexity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compliance</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Cost</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Benefit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As already described, DL uses inference to provide five different types of reasoning: subsumption, instance checking, relation checking, concept consistency, and knowledge base consistency. These can be used to analyze the entity and relationships types specified in the conceptual schema as well as the individual instances of these types. Table 1 shows the relationship between the different types of reasoning and the eight categories of model analysis proposed by [20].

The cells filled with an X indicate that the reasoning technique can be used to support a model analysis type. It is assumed that this support might be partial, with the full support probably requiring combination with other techniques. It should also be referred that the matching between analysis types and reasoning concerns analysis specifically at the model level, leaving out considerations concerning analysis at
the meta-model level, which are nonetheless relevant but considered out of the scope for this paper.

Subsumption and instance checking can be used to support the analysis of the heterogeneity of a model, as it involves classifying types and its individuals in order to refactor the model. Relation checking supports several types of model analysis since it determines how entity types relate to each other through a chain of relationship types. Thus it supports dependency analysis as it involves checking whether two elements are related with each other and determining the nature of the relationship. It also supports analyzing the coverage of a model, i.e. how the model’s artefacts intersect the architecture layers. It also plays a role in interface analysis since it enables assessing the degrees of cohesion and coupling of an artefact. Finally, relation checking partially aids the computation of metrics required for complexity analysis, cost analysis and benefit analysis. Concept consistency and knowledge base consistency can aid compliance assessment. Concept consistency can be used to verify the existence of contradictions between the definitions of a type. This can be used to determine whether rules or requirements are being met. Knowledge base consistency checking can be used to verify whether the definition of an individual entity or relationship complies with the definition of the corresponding type, i.e. whether a model is valid against the specification of a meta-model.

Nevertheless, reasoning is not able to fully support all categories of model analysis. In particular, the application of reasoning to dependency analysis is limited to the relationships between different types since it is not possible to analyze relationships between type properties. Reasoning is also not a good candidate to perform the computation of metrics required for complexity analysis and cost/benefit analysis. Thus, description logics based reasoning requires to be complemented with other analytical techniques. Moreover, the suitability of using reasoning also needs to be studied in terms of performance and scalability, especially when analyzing large enterprise architecture models with a dense graph of relationships between the entities. A technique that is readily available for further supporting analysis is the use of SPARQL which can be applied on top of an inferred model. This allows the use of aggregates, which can be used to perform simple calculations.

5. Analysis using Ontologies

Based on this federated arrangement, the different types of EA model analysis will be demonstrated and described below using an example EA description and a determined EA meta-model as the UO. The demonstration will then be based on the use of the ArchiMate modelling language and will use a fictitious running example scenario comprising several elements belonging to the different abstraction layers depicted in ArchiMate. Figure 2 depicts a layered view of the scenario ArchiMate model.

To create the UO, 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

Figure 2. Scenario ArchiMate model
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 individuals of the classes existing in the ArchiMate ontology created in the two previous steps, which correspond to the elements modelled in an ArchiMate model.

The whole process was aided by the existence of an XML representation for ArchiMate models in the Archi2 modelling tool. Hence, an XML-to-OWL representation transformation tool was created for automating the aforementioned steps, allowing for the direct conversion of the models produced in the Archi tool to OWL. The scenario OWL file produced by the conversion can be found at http://sysresearch.org/ontologies/scenarios/demo_scenario.owl and the ArchiMate meta-model represented in OWL can be found at http://sysresearch.org/ontologies/UO.owl.

5.1. Dependency Analysis

Taking advantage of the possibility of introducing semantics on the models, 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.

Figure 3. Dependency analysis using DL

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 3 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 4 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 4. Reasoner explanation for dependency

The use of SPARQL on top of the inferred model further allows the use of aggregates, as can be observed below. In this case, the number of technological dependencies of the aforementioned service can be computed using the query below, which makes use of the aggregate COUNT which counts on how many elements of the type System Software, Node, Device, and Infrastructure Service the individual labeled “Business_Service_1” depends upon. The execution of the query returns the result “6”.

```
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#>

SELECT (COUNT(?subject) as ?count)
WHERE {
  {?subject rdf:type uo:SystemSoftware}
  UNION {?subject rdf:type uo:Node}
  UNION {?subject rdf:type uo:Device}
  UNION {?subject rdf:type uo:InfrastructureService}.
  ?object rdfs:label "Business_Service_1"^^xsd:string.
}
```

5.2. 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.

```
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:dependsDown ?object} .
}
```

However, SPARQL can be used to this end. The SPARQL query above 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, in this case, if there are any individuals of the type Business Process or Application Component upon which a determined Business Process depends upon. 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”.

5.3. 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.

```
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 {
}
```

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. It matches all individuals with inward or outward relationships with the individual labeled “Business_Process_6”. Figure 5 reveals the results of the executed query on top of the non-inferred model.

5.4. 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.

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

SELECT DISTINCT ?class ?individual WHERE {
  ?class a owl:Class.
  ?ind rdf:type ?class.
  ?class rdfs:subClassOf uo:ArchimateConcept
}
```

Nonetheless, the use of SPARQL can provide interesting insights in a tabular format. The SPARQL query above can be used to retrieve all the individuals...
per class of the scenario being addressed. Figure 6 depicts an excerpt of the results to the query.

5.5. 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. It makes use of the COUNT aggregate on top of triples relating individuals to their respective type.

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX uo: <http://sysresearch.org/ontologies/UO.owl#>
SELECT ?class (COUNT(?individual) as ?count)
WHERE {
  ?class a owl:Class.
  ?class rdfs:subClassOf uo:ArchimateConcept.
  ?individual rdf:type ?class.
}
GROUP BY ?class ORDER BY DESC(COUNT(*))
```

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. Making use of the COUNT aggregate, the query returns, for each individual of the type Application Component, the relationship types it is engaged in, and the number of occurrences of these relationships. Figure 7 depicts an excerpt of the results of the execution of the query above.

```sparql
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 rdfs:subPropertyOf uo:dependsUp} UNION
  {?predicate rdfs:subPropertyOf uo:dependsDown}.
}
GROUP BY ?instance1 ORDER BY DESC(COUNT(*))
```

Moreover, the query can be modified to target a specific type of relationship, in this case, the dependencies defined in Section 5.1, as can be seen above. While this example still focuses in individuals of the type Application Component, it focuses solely on relationships of the type dependsUp and dependsDown. This allows retrieving the total number of dependencies from and to individuals of the aforementioned type.

5.6. 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, a DSO specializing the concept of Infrastructure Service can be integrated with the UO, 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 individual AvailabilityRequirement with the following type:

```
Requirement and (realizedBy only InfrastructureService and (hasAvailability 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 8 depicts an inconsistency being detected by the HermiT reasoner and explained in Protégé.

Figure 8. Compliance analysis inconsistency explanation

5.7. Cost Analysis

The demonstration of this type of analysis involves augmenting the scenario’s architecture description with cost related information. In this case, this can be achieved with the integration of a DSO that adds cost information to the infrastructure nodes modeled in the scenario through a data property `cost` with a data range of the type double. Table 2 depicts the associated cost information associated with each node in the DSO.

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

The addition of this information coupled with dependency information on top of the inferred ontology makes possible the execution of interesting 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 SPARQL query above, which makes use of the aggregate SUM. It should be noted that the query matches triples where the predicate in the `dependsUp` object property, meaning that dependency information is being used for the calculation of the cost values. The results of the query can be observed in Figure 9.

Figure 9. Cost analysis SPARQL query results

5.8. 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 10 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 determined business processes have the income values displayed in Table 3.

<table>
<thead>
<tr>
<th>Business Process</th>
<th>Income</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Process 1</td>
<td>12.0</td>
</tr>
<tr>
<td>Business Process 2</td>
<td>7.5</td>
</tr>
<tr>
<td>Business Process 3</td>
<td>3.1</td>
</tr>
<tr>
<td>Business Process 8</td>
<td>16.3</td>
</tr>
</tbody>
</table>

Using the inferred model, the income generated by business processes realizing Goal 6 can be verified with the SPARQL query below which, similarly to the cost example, takes advantage of dependency information on top of the inferred model.
SELECT ?label1 ?income
WHERE {
}

ASK
WHERE {
  demo_scenario:Goal_6 demo_scenario:income ?goalinc .
  bind((?income * 100)/?goalinc as ?percentage)
}
GROUP BY ?INCOME ?goalinc
HAVING (SUM(?income) >= ?goalinc)

In order to determine if Goal 6 is being met, a SPARQL query in the ASK form can be elaborated as displayed above. 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. 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 11 depicts the resulting percentages for the example.

6. Conclusions

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.

Figure 11. Benefit analysis SPARQL query results.
7. Acknowledgements

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).

8. References


