Deliverable Nr: D.A3.3

Semantic Annotation language and tool for Information and Business Processes

Work Package – A3

Leading Partner: CNR-IASI

Security Classification: Project Participants (PP)

February 14, 2006

Version 1.0
Table of contents

APPENDIXES

1 EXECUTIVE SUMMARY  1
1.1 Structure of the deliverable  2
1.2 Disclaimer  3

2 INTRODUCTION
2.1 Positioning  4
2.2 Purpose of A*  5

3 REQUIREMENTS
3.1 Categories of requirements  6
3.2 Requirements related to the kind of annotation  6
3.3 Requirements related to the deployment and distribution  7

4 METHODOLOGY
4.1 Introduction to Semantic Annotation  8
4.2 Semantic mismatches  9
4.3 Four steps to a Full Semantic Annotation  10
4.4 The four levels Annotation Process in A*  12
4.4.1 Terminological Semantic Annotation (TSA)  12
4.4.2 Path Semantic Annotation (PSA)  13
4.4.3 Simple Semantic Annotation (SSA)  14
4.4.4 Full Semantic Annotation (FSA)  15
4.5 Multilevel Semantic Annotation Methodology  17
4.5.1 The Multilevel Approach Formalisation  17
4.5.2 Summary of annotation methodology  19

5 FUNCTIONAL SPECIFICATIONS
5.1 Functions related to annotation characteristics  21
5.2 Functions related to deployment and distribution  21

6 ARCHITECTURE
6.1 Semantic suite repositories  23
6.2 A* Architectural schema  24
6.3 Main components of A*
6.3.1 The A* GUI  25
6.3.2 The Application tier  26
6.3.3 OntoLoader and Ontology Graphical Visualizer (OntoViz)  29
6.3.4 A* Application Logic  30
6.4 Annotation repository and the Annotation repository service
6.4.1 The Annotation Repository  30
6.4.2 The Annotation Repository service  32

7 CONCLUSIONS  33

8 REFERENCES  34
Table of Figures

Figure 1 - Athena Semantic Suite Components ...................................................................................... 4
Figure 2 – Examples of paths from the Reference Ontology, Paths(RO) .................................................. 18
Figure 3 – Multilevel Semantic Annotation ................................................................................................ 20
Figure 4 – Multilevel Semantic Annotation and Mismatches ..................................................................... 20
Figure 5 – A3 macro architecture ............................................................................................................. 23
Figure 6 – A* macro architecture ............................................................................................................. 25
Figure 7 – A* GUI: the main window ........................................................................................................ 26
Figure 8 - Client Manager ......................................................................................................................... 27
Figure 9 – Resource document and ontology acquisition and visualization ............................................. 29
Figure 10 - Relationships between the Annotation Repositories ............................................................... 30
### Appendixes

<table>
<thead>
<tr>
<th>No</th>
<th>File name</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>AppendixA ResViz - V1.0.doc</td>
<td>Appendix A: ResViz</td>
</tr>
<tr>
<td>B</td>
<td>AppendixB OntoViz - V1.0.doc</td>
<td>Appendix B: OntoViz</td>
</tr>
<tr>
<td>C</td>
<td>AppendixC Ontology import - V1.0.doc</td>
<td>Appendix C: Ontology import</td>
</tr>
<tr>
<td>D</td>
<td>AppendixD Examples of annotation - V1.0.doc</td>
<td>Appendix D: Examples of annotation</td>
</tr>
<tr>
<td>E</td>
<td>AppendixE Annotation Repository - V1.0.doc</td>
<td>Appendix E: Annotation Repository</td>
</tr>
<tr>
<td>F</td>
<td>AppendixF User manual - V1.0.doc</td>
<td>AppendixF User manual</td>
</tr>
<tr>
<td>G</td>
<td>AppendixG Instalation Guide - V1.0.doc</td>
<td>AppendixG Instalation Guide</td>
</tr>
<tr>
<td>H</td>
<td>AppendixH An approach for integration of A3 solutions and A6 models repository - V1.0.doc</td>
<td>AppendixH An approach for integration of A3 solutions and A6 models repository</td>
</tr>
</tbody>
</table>
1 Executive summary

This deliverable presents a methodology for semantic annotation of digital resources and a tool, A*, that implements a semantic annotation environment based on such a methodology. The proposed semantic annotation tool has been conceived for a large variety of digital resources (e.g., business documents, business processes, web services) involved in a typical cooperation between enterprises in an e-Business scenario. Semantic annotation has the first goal of checking the alignment of a digital resource content with a given Reference Ontology (RO). Then, other semantic services based on the semantic annotation are: reconciliation of business documents exchanged during the execution of a business process; advanced semantic search of digital resources; consistency verification of a set of resources; verification of subsumption (i.e., conceptual refinement), equivalence, disjointness of (annotated) resources (i.e., semantic incompatibility).

Semantic annotations will reveal their predisposition to carry out cooperative tasks in the community that has expressed the RO. In semantic interoperability, ontologies play a central role. An ontology represents a shared, agreed conceptualization of a given application domain. In an e-Business scenario, it can be seen as the semantic representation of a business sector where players cooperate and compete for business, where the most common activity is buying and selling goods and services.

An annotation is essentially a mapping between the content of a business digital resource and the reference ontology. The proposed method considers the content and the structure of the documents to be exchanged and contrasts them with the ontology, with the goal of building a mapping to the RO. Semantic annotation is a complex task; to make it easier the proposed method is based on an incremental approach that, starting from simple keyword based annotations, evolves towards more advanced and rigorous ones, reaching the last level where the annotation is represented by OWL expressions.

In the second part of the document, the deliverable describes also the functionalities and the technical architecture of the A* semantic annotation tool, which has been developed, and is released, as part of the Athena Semantic Framework by the Project A3. The A* tool supports the user to progressively identify possible mappings between the digital resource and the RO specifically selected. To this end, A* is characterised by a friendly interface capable of automatically proposing matching RO elements starting from the analysis of the elements of the business resource.

1.1 Structure of the deliverable

The present deliverable is organized in the following way:

Chapter 2: is an introduction to the Athena Semantic Framework and the position of the A* tool within this suite

Chapter 3: describes the functional and technical requirements of A*. Such requirements have been identified in the analysis of the Athena semantic needs, focusing in particular on semantic mediation solutions required by the Athena scenarios. The semantic solutions have been also conceived having in mind the technical architectures that have been developed by the different Action Line projects that constitute the Athena Interoperability target environments in which A* will operate.

Chapter 4: is a description of the methodologies that have been considered as the methodological foundation of the proposed semantic annotation solution.

Chapter 5: outlines the functional specifications that guided the implementation of A*, according to the requirements described in Chapter 3.

Chapter 6: describes the architecture and the components of the A* tool and its relationships and interfaces with the other tools composing the Athena Semantic Framework.

Chapter 7: describes the main features of the Zope platform that has been selected for the development of the A* tool, as well as the Athos ontology management tool.

The general organization of this deliverable has been conceived with the aim to keep the number
of pages of the primary document within a limited size (typically, below 50). Then a number of appendixes have been attached. The appendixes gather specific topics that are an important outcomes of the activities of the task A3.3, but at the same time they can be elaborated in a precise manner in a dedicated text. In this way, we achieve a compact, highly legible primary document, but at the same time we provide appendixes where the key topics requiring further elaboration are again treated, at a right level of elaboration.

1.2 Disclaimer

A* V1.0 is an experimental prototype developed within the Athena project and as such is released within the Athena project for the uses that this project entails.

All questions and comments related to A* V1.0 should be addressed to osimi@iasi.cnr.it
2 Introduction

2.1 Positioning

The A* tool is part of what has been indicated as the Athena Semantic Framework (ASF) of Tools. The next figure shows a graphic representation of the Athena Semantic Framework of tools. The figure shows the use of ASF as a set of tools aimed at reconciling messages or achieving transformations between different messaging formats, and performing semantic searches of annotated digital resources, such as business documents or CBP components.

![Athena Semantic Suite Components](image)

*Figure 1 - Athena Semantic Suite Components*

The following systems are shown in the previous figure:

- **Athos**, the Athena Ontology Management System (OMS). An OMS allows the definition of both the domain concepts of an ontology and their relations; an ontology represents an agreed and shared understanding of a given domain.
- **A***, the Semantic annotation tool. Semantic annotation aims at giving a non ambiguous meaning to digital resources and represents a conceptual correspondence between resources and concepts in the ontology; the semantic annotation process results in semantic annotation expressions.
- **ARGOS**, the Reconciliation Rules generator: the semantic annotation expressions created using A* are fed into the ARGOS reconciliation rule generator, which enables the user to define forward and backward reconciliation rules; the reconciliation rules represent a procedural way of transforming ground resources (i.e., data) into ontology instances (forward transformation) and vice-versa (backward transformation).
- **ARES**, the runtime reconciliation engine: the actual reconciliation of the messages exchanged among two inter-operable business partners, is the result of the composition of a forward and a backward transformation of the content of message instances.
• **ASSERT**: the Athena Semantic Search Engine and Retrieval Technology platform allows the execution of Semantic queries against documents, process or model components retrieved from repositories of other AL A projects, once they have been semantically annotated with the A* tool. Queries are done against semantic annotation expressions (held in the Semantic Annotation Repository). Original resources are retrieved since there is a link between annotation expressions and annotated resources.

• **THEMIS, the RDF models repository**: the different components of the Athena Semantic Suite share RDF models; a single repository for all those files, in order to manage them in a more integrated way, has been implemented. This component could include also functionalities for mapping XSD docs into RDFS and the reverse.

### 2.2 Purpose of A*

The main purpose of the task A3.3 is the development and delivery of A* the Athena Semantic annotation tool. Semantic Annotation (SA) is a methodology used to make explicit the semantic content of a resource in a formal way. This purpose is achieved by associating a resource with an expression based on the concepts and relationships defined in a domain reference ontology.

Semantic annotation allows for the creation of a sort of a semantic image of resources that can be used in several applications. In particular:

• **Knowledge management**: semantic annotation can be used for a wide range of content-oriented applications such as classification, retrieval, extraction, translation, presentation, and query-answering. The annotations allow classification of documents in semantic indexes that can be used to retrieve the information.

• **Semantic compatibility analysis**: SA can be used to give meaning to software application elements, annotating them with the concepts of a domain ontology; this allows to identify similarity and differences and to carry out the preliminary analysis to achieve interoperability.

• **Reconciliation rules generation**: starting from SA expressions of two cooperating software components, it is possible to analyze the information they exchange and, to a certain degree, their behavioral characteristics to produce a set of semantic reconciliation rules. Such rules aim at bridging the semantic discrepancies and to allow (in case of lossless mismatch) for a seamless cooperation.

In the ATHENA context A* allows to semantically enrich the whole knowledge of an enterprise (documents, business models, and software components). This topic should allow managing the different forms of enterprise knowledge in a unified environment in order to have reachable and reusable resources for enhancing the enterprise’s interoperability capabilities.
3 Requirements

3.1 Categories of requirements

In the definition of the characteristics that the Athena A* Semantic Annotation tool should possess two main categories of requirement characteristics have been defined; they are the following:

1. requirements related to the kind of annotation that can be performed on the objects to be annotated;
2. requirements related to the deployment and distribution of the tool and its interaction with both operators and the other tools that compose the Athena Interoperability solution.

These requirements are described in the following paragraphs.

3.2 Requirements related to the kind of annotation

Semantic annotation, and in general methods and tools for the annotation of digital resources, represent a very active research field since (at least) a decade. In analysing the results available in the literature and formulating the requirements that the appropriate Athena Semantic Annotation tool should satisfy, we decided to adopt a systematic approach, based on seven key features, such as the level of formality or the intended usage, that determine the kind of annotation that can be performed. Below, the seven features are illustrated.

- **Annotated resource.** The tool should be conceived for annotating different kinds of electronic document formats, with a special regard towards XML and resources description formats;
- **Intended users.** The produced annotation should be understandable both by a machine and a human being. Being understandable by a machine allows the annotation to be used by automatic applications like for instance, automatic reconciliation rules generation and document retrieval. Being understandable by a human being allows the annotation to be used, for instance, as modelling specifications.
- **Annotation formalism.** The semantic annotation can be a time consuming activity. It can be used for different purposes (i.e, simple or advanced search and retrieval of documents, generation of reconciliation rules), which cannot require the same level of formality and details. Furthermore, it can be performed by different actors (i.e., business or technological people) with different skills. For these reasons, the annotation methodology should provide different levels of annotation, possibly incremental respect to each others, in order to be suitable for different purposes, different actors.
- **Terminology restriction.** The annotation methodology should constrain the user to a restricted and shared vocabulary in order to have a common reference by which building the annotations. It implies two things:
  - The more this common reference is shared, the more the annotation can be widely understood and used
  - The more this common reference is structured and formally described, the more it can improve the formality of the annotation expressions and consequently the automatic use of it.
- **Positioning.** The linkage of the annotation to the annotated resource should be as less intrusive as possible.
- **Level of Abstraction.** The annotation methodology should allow to annotate information at both conceptual and instance level.
- **Methodological support.** The annotation tool should provide some automatic supports for helping the user in the definition of the annotation expressions.
3.3 Requirements related to the deployment and distribution

The following requirements were identified by the development team:

- **Web applications**: the semantic annotation tool should be accessible through a simple web browser from the end users.
- **User friendly interface**: the tool should provide a user friendly interface; this interface should hide, wherever possible the complexities of the semantic annotation, through different levels of annotation creation.
- **Graphic document navigation**: the user should be able to navigate on the document to be annotated, via a graphic representation of the document path.
- **Interface with modelling systems**: the tool should provide the possibility to import modelling documents in different formats. The priority of the formats to be imported is to be given to RDF(S) and XMI Light formats. The modelling systems should also be allowed to access, via web service the semantic annotation repository, to retrieve the generated annotation expressions, linked to their models.
- **Interface with other A3 tools**: appropriate interfaces should be provided between A* and Argos (the reconciliation rule generator) as well as between A* and Athos OMS.

In the section 5, *Functional specification*, the requirements that have been addressed in the definition of the annotation methodology and the design of the tool will be shown.
4 Methodology

4.1 Introduction to Semantic Annotation

This section illustrates the main features of the A* method for semantic annotation (SA) and checking of the alignment of a Document Schema (DS), with respect to a given Reference Ontology (RO). As anticipated, such a method is based on a stepwise semantic annotation approach, where the elements of a DS are associated with ontology-based expressions of an increasing complexity.

The method is organised in two phases. Firstly we have the diagnostic phase aimed at identifying the possible (terminological, structural, and semantic) mismatches between the DS and the RO. Secondly, we have the remedial phase, where the actual mapping is defined. When a mismatch is identified, the element of the document (ds-element) is associated with an expression representing the semantics of the element in terms of RO. This association is referred to as “mapping”. Unfortunately, not all the mappings are able to fully (and precisely) capture the intended semantics of the DS element. We will elaborate more on this issue. The semantic mapping is based on the construction of a semantic annotation (SA) expression.

SA is a well known technique that has been proposed in literature originally to annotate documents and web pages (see ATHENA WDA3.5, State of the Art on Ontology-based Semantic Annotation methods and tools). Recently, SA has attracted much attention in the context of the Semantic Web for its great potentiality in solving interoperability problems. Early proposals have seen SA embedded in HTML metadata elements (and, successively, in XML). The more recent proposals are aimed at the creation of autonomous structures that represent, in a formal, controlled way, the semantic content of a web resource (e.g., a document, a business process or an eService).

As anticipated, among typical applications of semantic annotation, we can find:

- **Semantic Search**, i.e., the possibility of retrieving digital resources (not only documents) on the basis of their semantic content. This service is at the basis of the following, more specific applications;
- **Knowledge Management**, for organization and retrieval of enterprise knowledge, in its diverse forms, such as business processes, organization models, best practices, products with their lifecycle models.
- **Web Services** publishing and discovery, with semantic matchmaking of requested and offered services; Web service composition, with the possibility of verifying reciprocal compatibility.
- **Semantic Interoperability**, by annotating local software application resources (information and processes) to support business cooperation among enterprise systems.

The solution presented in this document is positioned within the last application area: interoperability among enterprise software applications (ESA). In particular we will focus on documents that are exchanged (typically as payload of SOAP messages) among software applications, to assess their alignment with respect to a given reference ontology, with the aim of identifying the semantic mismatches.

As anticipated, the proposed method is based an incremental approach, with four different levels of annotation of a document schema, each of which allows a given set of mismatches to be singled out. The progression of the annotation levels requires an increasing level of precision.
4.2 Semantic mismatches

In our work we identified a limited number of possible mismatches. Such mismatches categories are sketchily recapped in the table below [Table 1]. The examples reported in the table are drawn from an eProcurement business scenario proposed in the Athena project and aim at representing the different cases in an informal, intuitive way (see Appendix D1, FSA Example for a complete, formal account). In particular, we consider a PurchaseOrder business document and a corresponding reference ontology. In our example we assume, that the Document Schema is represented in the RDFS1/N3 notation and the Reference Ontology in OWL/N3 notation.

As shown in the table, the possible mismatches have been divided into two broad categories: lossless and lossy mismatches. Lossless mismatches are cases in which annotation can fully capture the intended semantics of the annotated element, while lossy mismatches represent cases where a semantic preserving2 mapping to the reference ontology cannot be built.

<table>
<thead>
<tr>
<th>Lossless mismatches</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>Naming</td>
<td>different labels for the same concept</td>
</tr>
<tr>
<td>Attribute Granularity</td>
<td>the same information is decomposed into a different number of attributes (or subattributes)</td>
</tr>
<tr>
<td>Structuring</td>
<td>different design structures in organising the information</td>
</tr>
<tr>
<td>SubClass-Attribute</td>
<td>an attribute, with a predefined value set, is represented by a set of subclasses, one for each value</td>
</tr>
<tr>
<td>Schema-Instance</td>
<td>data holding schema information. In the DS example, the semantics of the second field (Name) depends on the value of the first field (Role). In the RO, the semantics is fully captured in the attribute names.</td>
</tr>
<tr>
<td>Encoding</td>
<td>different formats of data or units of measure (a Weight expressed in different unit of measure)</td>
</tr>
</tbody>
</table>

1 without loss of generality, we assume a lossless transformation of a document from, say, XSD to RDFS
2 A formal treatment of the semantics falls outside the scope of the paper. Here the semantics is addressed at an intuitive level.
### Lossy mismatches

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Document Schema</th>
<th>Reference Ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content</td>
<td>different content denoted by the same concept - typically expressed by enumeration (the concept described by different enumeration items)</td>
<td>PaymentTerms=(30 days, 60days, 90days)</td>
<td>PaymentTerms=(30 days, 45days)</td>
</tr>
<tr>
<td>Coverage</td>
<td>presence/absence of information (a given information is considered in the RO, but not in DS)</td>
<td>preferredDeliveryDate is not considered in the DS</td>
<td>preferred delivery date is present in the RO</td>
</tr>
<tr>
<td>Precision</td>
<td>the accuracy of information</td>
<td>Size of a pallet expressed by three integer values: height, length, width</td>
<td>Size of a pallet expressed by a constant conventional value: (small, medium, large)</td>
</tr>
<tr>
<td>Abstraction</td>
<td>level of specialisation/refinement of the information: generic vs detailed representation</td>
<td>DeliveryTerms</td>
<td>DeliveryTerms gets specialized into: NationalDeliveryTerms and InternationalDeliveryTerms</td>
</tr>
</tbody>
</table>

Table 1. Sorts of mismatches

### 4.3 Four steps to a Full Semantic Annotation

The proposed method for semantic annotation is based on a stepwise approach that allows an annotation to be progressively refined. It starts from the simplest form, a vector of keywords, and progressively evolves, through 4 different stages, till it reaches the last level, represented by an OWL expression. Here we briefly introduce the four steps of our approach, and we indicate the sorts of mismatches that can be identified at each step. In the next section, the four steps will be extensively illustrated.

#### Step 1 – Terminological Semantic Annotation Vector (TSAV)

In this step the terms used in the DS are contrasted with the terminology of the RO (i.e. the "rdf:id" and the "rdf:label" used for the ontology elements) to find terminological matches. The annotation is then achieved in the form of vectors of RO terms. In particular, the reported ontology terms correspond to the concepts preferred terms, that are reported under the "id" element in the ontology (unless otherwise indicated by the user).

During this step, the following mismatches are singled out:

- **Naming mismatch**
- **Coverage mismatch**
- **Abstraction mismatch**

#### Step 2 – Path Semantic Annotation Vector (PSAV)

In this step, the terms identified in the TSA are used to achieve a first annotation that considers the structures of both: document and ontology. Therefore, terms are composed to form paths. Here we consider in particular property paths, i.e., that paths that start from an entity and connect it to one of its
atomic properties (i.e., typed with a basic type: integer, string, etc.) For instance,

Order.has_orderHeader.orderHeader.has_buyerInfo.buyerInfo.buyerInfo_has_organisationInfo.organisationInfo.has_phone:string is a property path.

We focus on property paths since atomic properties are the key elements used in data exchange (i.e., they actually carry data). But another important issue is that a single term is often not sufficient to unambiguously express the intended semantics. For instance, the property has_phone may represent information associated to Buyer_BusinessActor or to SellerBusinessActor depending on the paths in which it participates.

Each document path is associated to one or more RO paths that represent the PSA. The matching of the single elements of the paths is performed by considering the mappings achieved in the Step 1.

In this step, the following kind of semantic mismatches are identified:

- **Structural mismatch.**
- **SubClass-Attribute mismatch**
- **Schema-Instance mismatch**
- **Content mismatch.**

In the first two steps the method produces semantic annotations represented by vectors of RO terms and RO paths (semantic annotation vector, sav) that are associated to individual terms and paths of the business document, respectively. As illustrated above, the use of semantic annotation vectors is useful to identify a significant number of semantic mismatches. However, we need to progress further in the refinement of the annotations, introducing semantic annotation expressions (sax) that make use of specific operators.

**Step 3 – Simple Semantic Annotation eXpression (SSAX)**

In this step, the ontology paths, identified in the previous step, are used to compose Simple expressions by using abstract operators. An SSA expression is typically a composition of RO paths that better captures the intended meaning of the annotated DS element. This level has the specific aim to facilitate the reconciliation phase.

In this step, the following kinds of semantic mismatches are identified:

- **Attribute Granularity mismatch.**
- **Encoding mismatch**
- **Precision mismatch.**

**Step 4 – Full Semantic Annotation eXpression (FSAX)**

In this step, a simple annotation is refined and coded into OWL. Every path that appears in the simple annotation expression is translated into OWL DL code in automatic way. In this way, we have specified the annotation expression, in a formally grounded ontology representation language.

It is important to repeat that each step refines the precision of the annotation of the previous level. Every Semantic Annotation assumes the following form:

\[ ds-elem =: sav | sax \]

in case the semantic annotation vector (sav) or the semantic annotation expression (sax) fully
captures the intended meaning of the annotated element. Otherwise, we have the following situations (and the corresponding connectives):

- $ds\text{-}elem \triangleright: sav | sax$ (Overspecification), to express that the annotated $ds\text{-}elem$ has a richer intended semantics than the associated annotation expression or vector;
- $ds\text{-}elem \triangleleft: sav | sax$ (Underspecification), to express that the annotated $ds\text{-}elem$ has an intended semantics poorer than the associated annotation expression or vector;
- $ds\text{-}elem partOf sav | sax$, to express that a $ds\text{-}elem$ represents a component of the structured concept identified by the annotation expression or vector.

4.4 The four levels Annotation Process in A*

The semantic annotation process represents a first phase in which a given legacy system is confronted with its inherent inclination to interoperate within a given business community. In fact, the reference ontology RO is assumed to be a proper representation of the business domain and, in particular, of the part that will be involved in the networked business activities. It is fair to assume that, if one cannot find in the ontology a concept (and the corresponding terms) representing a given information or service, such a concept is outside of the scope of the community.

To build a formal annotation expression two basic competences are necessary: (i) a competence on the application domain, and (ii) a competence on the formal knowledge representation language used for the RO and the semantic expression. Such competencies typically belong to different kind of experts. The proposed multi-level semantic annotation method has been conceived to allow different experts to both contribute to the annotation process.

Below we elaborate on the four levels that have been presented in the previous section:

- Terminology Semantic Annotation;
- Path Semantic Annotation;
- Simple Semantic Annotation;
- Full Semantic Annotation.

The first two produce annotation vectors ($sav$) while the second two annotation expressions ($sax$).

4.4.1 Terminological Semantic Annotation (TSA)

At this level a SA is represented by a vector of terms belonging to the definition of concepts in the RO.

In the A* tool the identification of such terms is performed in a semi-automatic way. A string matching function comparing DS elements and RO terminology proposes an initial mapping, highlighting the possible mismatches, i.e., terms in DS that do not appear in the RO. The user has the possibility to confirm, reject, and/or introduce new terms (selecting them from the ontology).

When we talk about RO terminology, we refer to the labels of concepts, labels of relations, and their synonyms (that are generally specified in the concept structure of the ontology, in the field “terminology”). At this level the ontology is essentially used as a thesaurus.

In Table 2 we give an excerpt of the result of the TSA step for our example. In the first column the DS terms are indicated; in the second column we have the corresponding RO terms: terms in bold represent the ones suggested by the tool and accepted by the user, terms in italic represent the ones suggested by the tool but rejected by the user and terms underlined represent the terms added by the user. The third column reports the kind of mismatch raised.
As result of this phase, a TSA vector (tsav), built starting from terms in the ontology, is associated to each DS element. For instance, a tsav associated, through the connective "=:", to the ds-elem named

\[
\text{has\_name}
\]

is

\[
[\text{has\_Name, has\_PersonFirstName, has\_PersonSurname}].
\]

### 4.4.2 Path Semantic Annotation (PSA)

The PSA represents a first refinement of the annotation performed at the terminological level.

<table>
<thead>
<tr>
<th>DS - elem</th>
<th>TSA Vector</th>
<th>Mismatch</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>has_PostalAddress, has_emailAddress, PostalAddress_ComplexAttribute</td>
<td>Naming</td>
</tr>
<tr>
<td>addrDetails</td>
<td>has_PostalAddress, has_emailAddress, PostalAddress_ComplexAttribute</td>
<td>Naming</td>
</tr>
<tr>
<td>amount</td>
<td></td>
<td>Coverage</td>
</tr>
<tr>
<td>buyerInfo</td>
<td>Buyer_BusinessActor [purchaser, purchasing Unit, PU], relTo_Buyer</td>
<td>Naming</td>
</tr>
<tr>
<td>buyerInfo_has_organisationInfo</td>
<td></td>
<td>Coverage</td>
</tr>
<tr>
<td>Has_identifier</td>
<td>has_PartyTaxID</td>
<td>Naming</td>
</tr>
<tr>
<td>contactPerson</td>
<td>ContactPerson_BusinessActor</td>
<td>Naming</td>
</tr>
<tr>
<td>dimension</td>
<td></td>
<td>Coverage</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>has_contactPerson</td>
<td>relTo_ContactPerson, ContactPerson_BusinessActor</td>
<td>Naming</td>
</tr>
<tr>
<td>has_height</td>
<td>has_Height, [base to top measure, tallness, vertical measure, elevation, highness]</td>
<td>Naming</td>
</tr>
<tr>
<td>has_name</td>
<td>has_PersonFirstName, has_PersonSurname, has_part_StreetName, has_part_CountryName</td>
<td>Naming, Attribute granularity</td>
</tr>
<tr>
<td>has_transportationTerms</td>
<td>has_TransportationTerms</td>
<td></td>
</tr>
<tr>
<td>has_URL</td>
<td>has_URI</td>
<td>Naming</td>
</tr>
<tr>
<td>orderDate</td>
<td>has_IssueDate</td>
<td>Naming</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Examples of TSA
In the perspective of data exchange and reconciliation, the most relevant DS elements are the properties that at run-time carry the actual values. In general, the name of a property may not carry sufficient semantics to be unambiguously annotated. For instance, the property *date* has a different meaning if associated to an *invoice* (i.e., billing date) or to a *delivery note* (i.e., delivery date). To disambiguate such cases it is necessary to consider the full property path starting at the class and ending at the property.

Both the *ds-path* and the ontology paths in the PSA vectors (PSAV) are expressed with a dot notation. The *ds-path* to annotate is defined by the user selecting the terms that compose it using the A* interface. While the candidate ontology paths will be automatically proposed to the user, he has the possibility to confirm, reject, and/or introduce new paths. Each path is built combining the terms selected in the previous phase, taking into account the constraints imposed by the domain and range of the properties in the ontology. An ontology path is a dotted sequence of name of the concepts and relation in which every name of concept is followed by a name of relation. In the concatenation of concept and relation names the following constraints must be satisfied:

- the sequence must begin with a name of concept,
- the relation name can follow the name of the concept only if the concept is included in the domain of the property,
- the sequence ends with the name of the concept if the property links two concept,
- the sequence ends with a colon followed by the basic type if the property links a concept to a literal.

In the following example we will give a formal account of this intuitive notion of ontology path.

The candidate matching paths, proposed by A*, can be further elaborated by the user, who should have a certain knowledge of the ontology and its structure.

For instance, consider the *ds-path*:

```
buyerInfo.buyerInfo_has_organisationInfo.organisationInfo.has_contactPerson.contactPerson.has_name:string (1)
```

In building the PSA vector, we start considering the TSA produced in the previous step. The TSA vectors of the terms involved in this path are:

- TSA(*buyerInfo*) =: \[Buyer_BusinessActor\]
- TSA(*buyerInfo_has_organisationInfo*) =: \[Buyer_BusinessActor, Organization_BusinessActor\]
- TSA(*organisationInfo*) =: \[Organization_BusinessActor\]
- TSA(*has_contactPerson*) =: \[relTo_ContactPerson\]
- TSA(*contactPerson*) =: \[relTo_ContactPerson\]
- TSA(*has_name*) =: \[has_PersonFirstName, has_PersonSurname\]

Therefore, the system proposes that (1) is associated with following PSA vector:

```
[Buyer_BusinessActor.relTo_ContactPerson.ContactPerson_BusinessActor.has_PersonFirstName:STRING, Buyer_BusinessActor.relTo_ContactPerson.ContactPerson_BusinessActor.has_PersonSurname:STRING]
```

### 4.4.3 Simple Semantic Annotation (SSA)

At this point we can use the PSA vectors to build a first approximation of the semantic annotation.

---

**In the perspective of data exchange and reconciliation, the most relevant DS elements are the properties that at run-time carry the actual values. In general, the name of a property may not carry sufficient semantics to be unambiguously annotated. For instance, the property *date* has a different meaning if associated to an *invoice* (i.e., billing date) or to a *delivery note* (i.e., delivery date). To disambiguate such cases it is necessary to consider the full property path starting at the class and ending at the property.**

Both the *ds-path* and the ontology paths in the PSA vectors (PSAV) are expressed with a dot notation. The *ds-path* to annotate is defined by the user selecting the terms that compose it using the A* interface. While the candidate ontology paths will be automatically proposed to the user, he has the possibility to confirm, reject, and/or introduce new paths. Each path is built combining the terms selected in the previous phase, taking into account the constraints imposed by the domain and range of the properties in the ontology. An ontology path is a dotted sequence of name of the concepts and relation in which every name of concept is followed by a name of relation. In the concatenation of concept and relation names the following constraints must be satisfied:

- the sequence must begin with a name of concept,
- the relation name can follow the name of the concept only if the concept is included in the domain of the property,
- the sequence ends with the name of the concept if the property links two concept,
- the sequence ends with a colon followed by the basic type if the property links a concept to a literal.

In the following example we will give a formal account of this intuitive notion of ontology path.

The candidate matching paths, proposed by A*, can be further elaborated by the user, who should have a certain knowledge of the ontology and its structure.

For instance, consider the *ds-path*:

```
buyerInfo.buyerInfo_has_organisationInfo.organisationInfo.has_contactPerson.contactPerson.has_name:string (1)
```

In building the PSA vector, we start considering the TSA produced in the previous step. The TSA vectors of the terms involved in this path are:

- TSA(*buyerInfo*) =: \[Buyer_BusinessActor\]
- TSA(*buyerInfo_has_organisationInfo*) =: \[Buyer_BusinessActor, Organization_BusinessActor\]
- TSA(*organisationInfo*) =: \[Organization_BusinessActor\]
- TSA(*has_contactPerson*) =: \[relTo_ContactPerson\]
- TSA(*contactPerson*) =: \[relTo_ContactPerson\]
- TSA(*has_name*) =: \[has_PersonFirstName, has_PersonSurname\]

Therefore, the system proposes that (1) is associated with following PSA vector:

```
[Buyer_BusinessActor.relTo_ContactPerson.ContactPerson_BusinessActor.has_PersonFirstName:STRING, Buyer_BusinessActor.relTo_ContactPerson.ContactPerson_BusinessActor.has_PersonSurname:STRING]
```

### 4.4.3 Simple Semantic Annotation (SSA)

At this point we can use the PSA vectors to build a first approximation of the semantic annotation.
expression. Such an expression will make use of abstract operations, leaving to a successive moment the actual specification of the concrete ones. In this step, aiming at a simplified expression, we will use only two abstract operations:

**Abstract binary operator** ⊕. Such an operator says that the semantics associated to a *ds-elem* is obtained by combining two or more *on-elem*. Therefore, an SSA expression is built on PSA vectors previously identified. This is the case of the *Structure Organisation* mismatch. The abstract ⊕ operator is considered a sort of "place holder". When the actual data transformation and reconciliation are addressed, a computable operation needs to be indicated (i.e., concatenation/splitting of elements).

Below we show the SSA expression associated to the *ds-elem*:

```
buyerInfo.buyerInfo_has_organisationInfo.organisationInfo.has_contactPerson.contactPerson.has_name:string
```

The expression applies the ⊕ infix operator between two *on-paths* identified at the PSA level.

```
Buyer_BusinessActor.relTo_ContactPerson.ContactPerson_BusinessActor.has_PersonFirstName:STRING ⊕
Buyer_BusinessActor.relTo_ContactPerson.ContactPerson_BusinessActor.has_PersonSurname:STRING
```

**Abstract unary operator** φ. A further refinement with respect to the PSA level is the application of a generic function to a single path identified at the previous level (i.e., necessary in the case of *Encoding* and *Precision* mismatches). The φ operator expresses that, in the case of data transformation and reconciliation, a particular conversion needs to be applied (e.g., a conversion between two different units of measure).

For instance, in establishing the following semantic equivalence:

```
celsiusTemp =: φ farenTemp
```

the φ function is kept as a place holder for future use, such as in the data transformation and reconciliation, where it will be replaced by the required computation.

### 4.4.4 Full Semantic Annotation (FSA)

At this level, an FSA expression is modeled in OWL-DL, by using the OWL operators such as: `unionOf`, `IntersectionOf`, `AllValuesFrom`, `someValuesFrom`, `minCardinality`, `maxCardinality`.

In particular, at this stage the proposed approach is mainly concentrated on the translation of annotations defined at second level (PSA). This choice is due to the expressive power of OWL. In fact, PSA (Path Simple Annotation) annotations are represented by a vector of paths from the ontology, while SSA, the third level, also involves abstract operators (⊕, φ). Respect to that, OWL DL is not able to represent such operators.

In the following of this section, the translation of annotations at PSA level is outlined. Nevertheless, at the end of this section, even if the current OWL constructs are not able to express the semantics of operators involved at SSA level, a possible way to translate SSA annotations without completely losing the operators is briefly argued.

A fundamental requirement to effect this translation is that the reference ontology has been constructed under the constraint that for each relation in the RO it is possible to define the
corresponding inverse relation.

**Translation of PSA into OWL**

**Considered the previous PSA:**

\[ \text{buyerInfo.buyerInfo\_has\_organisationInfo.organisationInfo\_has\_contactPerson.contactPerson\_has\_name:string} = \]

\[ \text{[Buyer\_BusinessActor.relTo\_ContactPerson>ContactPerson\_BusinessActor\_has\_PersonFirstName:STRING, Buyer\_BusinessActor.relTo\_ContactPerson(ContactPerson\_BusinessActor\_has\_PersonSurname:STRING]} \]

- each path that compose PSA is translated in OWL DL as shown in the following:

  the ds-Path:
  \[ \text{Buyer\_BusinessActor.relTo\_ContactPerson\_BusinessActor\_has\_PersonFirstName:STRING} \]
  is translated into (for further details on the translation see Appendix D1, FSA Example):

  \[ \text{STRING } \sqcap \exists \text{ inverseOf\_has\_PersonFirstName} \]

  \[ \text{(ContactPerson\_BusinessActor } \sqcap \exists \text{ inverseOf\_relTo\_ContactPerson.Buyer\_BusinessActor)} \]

  the ds-Path :
  \[ \text{Buyer\_BusinessActor.relTo\_ContactPerson\_BusinessActor\_has\_PersonSurname:STRING} \]
  is translated into,

  \[ \text{STRING } \sqcap \exists \text{ inverseOf\_has\_PersonSurname} \]

  \[ \text{(ContactPerson\_BusinessActor } \sqcap \exists \text{ inverseOf\_relTo\_ContactPerson.Buyer\_BusinessActor)} \]

  Please note that for a better readability, the Description Logics SHIQ [1] syntax has been used to express the OWL code.

**Translation of SSA into OWL**

- each path that compose SSA is translated in OWL DL as shown above
- As previously said, using OWL DL is not possible to preserve completely the semantics of abstract operators (⊕, φ) involved in the SSA level. Nevertheless, a way to partially preserve this information is to use the owl:AnnotationProperty construct Appendix D1, FSA Example. Since this construct is conceived to add comments to OWL pieces of code, is not processable by any OWL reasoner.
4.5 Multilevel Semantic Annotation Methodology

In this section we present the formal background of the annotation methodology adopted in A*. Such a formal approach has been called Multilevel Semantic Annotation Methodology (MSAM) and is aimed at guiding the creation of the semantic annotation expressions.

It is important to note that the language used in the MSAM is characterized by a closed vocabulary. This means that, unlike other annotation languages, the user cannot define his/her own terms or named concepts in the annotations he/she creates. The sentences of SA language can be constructed only using the terms (i.e., concepts) defined in the Reference Ontology. A naming policy, inventing labels for variables, subroutines, relation names, etc., is one of the most critical aspects of the development of an information system. For this reason, one of the main characteristics of MSAM is the fact that, in building a semantic expression, the user needs to look at the ontology and can only select terms denoting defined concepts. Therefore the terminological elements of the ontology become part of the language; the generation of the annotation expressions is performed by a composition and/or transformation of ontology elements.

In this section we introduce the main formal issues of our layered approach to the semantic annotation, at an abstract (algebraic) level. In particular we describe the structures used in each of the four levels of annotation.

4.5.1 The Multilevel Approach Formalisation

Let RO={C, R} be the Reference Ontology where C is the set of all concepts and R=R_o [R_d where 
R_o is the set of relations between concepts and R_d (datatype property) is the set of relations between a concept and a base type.

Let T_RO=C'[R' be the RO terminology, where C' and R' are respectively the set of concept labels and the set of relation labels in the ontology.

Let dom(r) and ran(r) respectively be the domain and range of a relation r ∈ R.

Let G_RO be a direct Graph, G_RO = (N,A) where:
N = C
A = {(c_1, c_2) such that ∃ r R, c_1 2 dom(r) and c_2 2 ran(r) }
G_RO is a labeled graph: each node is identified by the label of the concept it represents, and each arc is identified by the label of the relation that originates it.

We define the RO property paths (ppaths(RO)) as the set of all the existing paths starting with a concept c and ending with a base type. Such a path is represented by a list of labels separated by dots, where the first element of the list is the label of the source concept and the last element is the label of an arc r2R_d.

The ppaths(RO) set is defined as follows:
ppaths(RO)={ (c'.r'_1,c'_1,r'_2,c'_2,...,r'_j,c'_j,...,r'_k-1,c'_k-1,r'_k) | c,c_2C, r_22R_d, r_1 2 R_o, where

8 ( r_i, r_{i+1}), ran(r_i)= c_i \subseteq dom(r_{i+1}) 1<i<k
and c', r'i are respectively the labels of c and r_i.}

Assuming that a DS, analogously to RO, can be represented as a direct Graph, where nodes are classes and arcs are properties in the RDF(S) document, we define the DS property paths (ppaths(DS)) as the set of all the existing paths starting with a class and ending with a base type. A ds-path is composed by a sequence of ds-elems analogously to what has been indicated for RO.

Figure 2 shows an example of an ontology, represented as a direct Graph, and its correspondent set of paths.

![Diagram](image)

Figure 2 – Examples of paths from the Reference Ontology, Paths(RO)

On the basis of the definitions given above we can define the following kinds of annotation expressions:

- a terminological semantic annotation vector of ds-elem, TSA(ds-elem) is represented as: a set of terms. In order to build a TSAV, the terms in the vector must be selected among the elements of the TRO set as follows:

  \[
  TSA(ds-elem) \subseteq TRO
  \]

- a path annotation vector of a ds-path, PSA(ds-path) is represented as:

  \[
  PSA(ds-path) = \{ p \in Paths(RO) \mid p = c_1.r_1.c_2.r_2.c_3,...,r_k.c_k \}
  \]

  where \( TRO = \{c_1, ..., c_t\} \)

  \[
  Paths(RO) = \{c_1.r_1.c_2.r_2.c_3,...,c_t.r_{t-1}.c_t \}
  \]

  \[
  c'_i = dom(r_i), \quad 1 < i < k
  \]

- a simple semantic annotation of a ds-path, SSA(ds-path) is an expression generated by the following grammar:

\[
SSA(ds-path) = \{ (c'_s,c'_t) \mid c_s = dom(r_i), c_t = dom(r_j), 1 < s < k, 1 < t < k \}
\]

where TSA(ds-path) = \{ ds-elem \in TRO \mid TSA(ds-elem) \}

\[
\]

where TSA(ds-path) = \{ ds-elem \in TRO \mid TSA(ds-elem) \}

\[
\]

where TSA(ds-path) = \{ ds-elem \in TRO \mid TSA(ds-elem) \}

\[
\]

where TSA(ds-path) = \{ ds-elem \in TRO \mid TSA(ds-elem) \}
G_{SSA} = <N, T, P, S>

N, non terminal symbols

T = PSA(ds-path) \[ \{\varphi, (,), \odot\}, \text{terminal symbols} \]

S, the starting symbol

P, production rules:

\[ S \rightarrow p \mid \varphi(S) \mid S \odot S \mid 2 \]

where \( p \in \text{PSA(ds-path)} \)

a full semantic annotation expression, is an OWL DL expression.

Let \( p = c_i . r_i' \ldots c_j . r_j' \ldots c_k . r_k' \) a path in an SSA, such a path is transformed into the following expression represented in the DL SHIQ syntax.

\[
\text{ran}(r_k) \cup \text{ran}(r_{k-1}) \ldots \text{ran}(r_1) \cup \text{ran}(c) \ldots \text{ran}(c_1)
\]

where \( r_i^{-1} \) is the inverse relation of \( r_i \) for \( 1 < k \)

The formalization of the \( \odot \) operator is not automatically derivable, and the operator \( \varphi \) is used as a support for the data reconciliation purpose.

4.5.2 Summary of annotation methodology

In Figure 3 a summary of the four levels of annotation is sketchily reported. For each annotation level, a brief description and the corresponding expert, able to build the annotation, are indicated. We distinguish three kinds of experts: (i) business person, with a sound domain knowledge but without technical (knowledge representation) competencies, (ii) domain expert, with a blend of domain knowledge and technical competencies, (iii) knowledge engineering, with full competences in ontology modelling, but with a limited domain knowledge.
The Annotation expression is a complex OWL DL expression. Ex: $FSA(\text{Order}.\text{has_orderHeader}.'\text{orderHeader}.\text{has_buyerOrderNumber}.'\text{buyerOrderNumber}.\text{has_identifier}.'\text{string}) =: \text{STRING} \cap \exists \text{inverseOf}._\text{has_PartyTaxID}.(\text{Buyer_BusinessActor} \cap \exists \text{inverseOf}._\text{relTo_Buyer}.\text{PurchaseOrder_BusinessObjectDocument})$

The Annotation expression is built on PSA previously identified and applying $\phi$ and $\ominus$ operators. Ex: $SSA(\text{buyerInfo}.\text{buyerInfo_has_organisationInfo}.'\text{organisationInfo}.\text{has_contactPerson}.'\text{contactPerson}.\text{has_name}.'\text{string}) >: \text{Buyer_BusinessActor}._\text{relTo_ContactPerson}.\text{ContactPerson_BusinessActor}.\text{has_PersonFirstName}.'\text{STRING}$ $\ominus$ $\text{Buyer_BusinessActor}._\text{relTo_ContactPerson}.\text{ContactPerson_BusinessActor}.\text{has_PersonFirstName}.'\text{STRING}$

The Path SA vector is built using terms previous identified. Ex: $PSA(\text{buyerInfo}.\text{buyerInfo_has_organisationInfo}.'\text{organisationInfo}.\text{has_contactPerson}.'\text{contactPerson}.\text{has_name}.'\text{string}) >: (\text{Buyer_BusinessActor}._\text{relTo_ContactPerson}.\text{ContactPerson_BusinessActor}.\text{has_PersonFirstName}.'\text{STRING}, \text{Buyer_BusinessActor}._\text{relTo_ContactPerson}.\text{ContactPerson_BusinessActor}.\text{has_PersonFirstName}.'\text{STRING})$

To annotate resource we use a vector of keywords (selected from the ontology terminology). Ex: $TSA(\text{buyerInfo}.'\text{has_organisationInfo}) =: [\text{Buyer_BusinessActor}, \text{Organization_BusinessActor}] ...$

According to the annotation levels introduced above, Figure 4 sketchily recaps the semantic mismatches illustrated in Section 2 and the level of annotation that allow each mismatch to be discovered and modelled.

```
<table>
<thead>
<tr>
<th>Levels</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full SA (OWL DL)</td>
<td>The Annotation expression is a complex OWL DL expression. Ex: $FSA(\text{Order}.\text{has_orderHeader}.'\text{orderHeader}.\text{has_buyerOrderNumber}.'\text{buyerOrderNumber}.\text{has_identifier}.'\text{string}) =: \text{STRING} \cap \exists \text{inverseOf}.<em>\text{has_PartyTaxID}.(\text{Buyer_BusinessActor} \cap \exists \text{inverseOf}.</em>\text{relTo_Buyer}.\text{PurchaseOrder_BusinessObjectDocument})$</td>
</tr>
<tr>
<td>Simple SA</td>
<td>The Annotation expression is built on PSA previously identified and applying $\phi$ and $\ominus$ operators. Ex: $SSA(\text{buyerInfo}.\text{buyerInfo_has_organisationInfo}.'\text{organisationInfo}.\text{has_contactPerson}.'\text{contactPerson}.\text{has_name}.'\text{string}) &gt;: \text{Buyer_BusinessActor}.<em>\text{relTo_ContactPerson}.\text{ContactPerson_BusinessActor}.\text{has_PersonFirstName}.'\text{STRING}$ $\ominus$ $\text{Buyer_BusinessActor}.</em>\text{relTo_ContactPerson}.\text{ContactPerson_BusinessActor}.\text{has_PersonFirstName}.'\text{STRING}$</td>
</tr>
<tr>
<td>Path SA</td>
<td>The Path SA vector is built using terms previous identified. Ex: $PSA(\text{buyerInfo}.\text{buyerInfo_has_organisationInfo}.'\text{organisationInfo}.\text{has_contactPerson}.'\text{contactPerson}.\text{has_name}.'\text{string}) &gt;: (\text{Buyer_BusinessActor}.<em>\text{relTo_ContactPerson}.\text{ContactPerson_BusinessActor}.\text{has_PersonFirstName}.'\text{STRING}, \text{Buyer_BusinessActor}.</em>\text{relTo_ContactPerson}.\text{ContactPerson_BusinessActor}.\text{has_PersonFirstName}.'\text{STRING})$</td>
</tr>
<tr>
<td>Terminology SA</td>
<td>To annotate resource we use a vector of keywords (selected from the ontology terminology). Ex: $TSA(\text{buyerInfo}.'\text{has_organisationInfo}) =: [\text{Buyer_BusinessActor}, \text{Organization_BusinessActor}] ...$</td>
</tr>
</tbody>
</table>
```

**Figure 3 – Multilevel Semantic Annotation**

<table>
<thead>
<tr>
<th>Users</th>
<th>Knowledge Engineering</th>
<th>Domain Expert/ Knowledge Engineering</th>
<th>Domain Expert</th>
<th>Business people</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Levels</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full SA (OWL DL)</td>
<td>The Annotation expression is a complex OWL DL expression. Ex: $FSA(\text{Order}.\text{has_orderHeader}.'\text{orderHeader}.\text{has_buyerOrderNumber}.'\text{buyerOrderNumber}.\text{has_identifier}.'\text{string}) =: \text{STRING} \cap \exists \text{inverseOf}.<em>\text{has_PartyTaxID}.(\text{Buyer_BusinessActor} \cap \exists \text{inverseOf}.</em>\text{relTo_Buyer}.\text{PurchaseOrder_BusinessObjectDocument})$</td>
</tr>
<tr>
<td>Simple SA</td>
<td>The Annotation expression is built on PSA previously identified and applying $\phi$ and $\ominus$ operators. Ex: $SSA(\text{buyerInfo}.\text{buyerInfo_has_organisationInfo}.'\text{organisationInfo}.\text{has_contactPerson}.'\text{contactPerson}.\text{has_name}.'\text{string}) &gt;: \text{Buyer_BusinessActor}.<em>\text{relTo_ContactPerson}.\text{ContactPerson_BusinessActor}.\text{has_PersonFirstName}.'\text{STRING}$ $\ominus$ $\text{Buyer_BusinessActor}.</em>\text{relTo_ContactPerson}.\text{ContactPerson_BusinessActor}.\text{has_PersonFirstName}.'\text{STRING}$</td>
</tr>
<tr>
<td>Path SA</td>
<td>The Path SA vector is built using terms previous identified. Ex: $PSA(\text{buyerInfo}.\text{buyerInfo_has_organisationInfo}.'\text{organisationInfo}.\text{has_contactPerson}.'\text{contactPerson}.\text{has_name}.'\text{string}) &gt;: (\text{Buyer_BusinessActor}.<em>\text{relTo_ContactPerson}.\text{ContactPerson_BusinessActor}.\text{has_PersonFirstName}.'\text{STRING}, \text{Buyer_BusinessActor}.</em>\text{relTo_ContactPerson}.\text{ContactPerson_BusinessActor}.\text{has_PersonFirstName}.'\text{STRING})$</td>
</tr>
<tr>
<td>Terminology SA</td>
<td>To annotate resource we use a vector of keywords (selected from the ontology terminology). Ex: $TSA(\text{buyerInfo}.'\text{has_organisationInfo}) =: [\text{Buyer_BusinessActor}, \text{Organization_BusinessActor}] ...$</td>
</tr>
</tbody>
</table>

**Figure 4 – Multilevel Semantic Annotation and Mismatches**
5 Functional specifications

The functions that have been developed in the Athena A* Semantic Annotation tool, as well as the corresponding requirements emerged in the analysis phase, are organised in two main categories; they are the following:

- functions related to the kind of annotation that can be performed on the objects to be annotated;
- functions related to the deployment and distribution of the tool and its interaction with both operators and the other tools that compose the Athena Interoperability solution.

These two groups of functions are described in the two following paragraphs.

5.1 Functions related to annotation characteristics

The A* functions that satisfy the requirements related to the kind of annotation characteristics, as outlined in the preceding Chapter 2 are the following:

- **Annotated resources** – A* has the capabilities to annotate a great variety of resources that are present in a business scenario: documents, web services, process and enterprise models, database schemata.
- **Intended consignee** – A* supports annotation expressions to be used both by the machine, for advanced knowledge processing, and human users. The latter are supported by a user-friendly interface.
- **Level of formality** – A* supports a multi-level annotation method that allows to start with a sketchy, informal annotation, represented by terminology vectors, and then to progressively enrich it to achieve the last formal level, represented by OWL expressions. Different levels of annotation can be achieved by different kinds of users: from business experts to knowledge engineers. Here we have a key feature of A* that cannot be found in existing proposals.
- **Terminology restrictions** – A* adopts the mandatory restriction for the terminology used in semantic annotation. Furthermore, it is ontology based, therefore the terms used in a SA must be selected from within the ones reported in the Reference Ontology;
- **Positioning** – in A* the annotation is attached to the annotated resource. The links from the annotation to the resource are implemented by XPointer/XPath expressions; such links need to be maintained when the resource is modified.
- **Level of abstraction** – A* allows for the annotation of instances starting from ontology concepts, by using the "instanceOf" link. This option gives more flexibility to the user, avoiding the proliferation of ontology instances.
- **Automatic user support** - in A* we adopt the progressive annotation, based on the layered approach. Such an approach is implemented in the user interface that guides the user from a simple keyword based annotation towards more complex levels. As anticipated, no other tool has implemented such a progressive method to support the user in the complex annotation task.

5.2 Functions related to deployment and distribution

The A* functions that satisfy the requirements related to the deployment and distribution, as outlined in the preceding Chapter 2, are the following:

- **Web applications**: A* tool will be accessible from the end users through a simple web browser.
- **User friendly interface**: A* is delivered with a user friendly interface; this interface hides the complexities of the semantic annotation, by providing guidance through different levels of annotation creation. Furthermore the user can create complex annotation expressions by successive links of simple annotations.
• **Graphic document navigation:** A* is provided with a Scalable Vector Graphic (SVG) representation of the document to be annotated; he/she is able to navigate on the document to be annotated, via graphic representations of the document paths.

• **Interface with modelling systems:** the tool provides the possibility of importing modelling documents in different formats. The presently supported formats are RDF(S) and XMI Light.

• **Web service interface:** modelling systems, as well as other applications are allowed to access, via web service the semantic annotation repository, to retrieve the generated annotation expressions, linked to their models or their documents.

• **Interface with other A3 tools:** appropriate interfaces are provided between A* and Argos (the reconciliation rule generator) as well as between A* and Athos OMS, through the development of specific A* components.

The above specifications have guided the implementation phase, from the functional point of view. In the next section the main architecture of the A* tool will be presented.
6 Architecture

6.1 Semantic suite repositories

Figure 5 reports a graphical representation of the functional bindings that exist among the different components of the Athena Semantic Suite.

In the figure, the main components of the A3 Semantic Suite are highlighted:

- **Athos**, the Ontology management system
- **A***, the Semantic annotation tool
- **Argos**, the reconciliation rules building tool
- **Ares**, the Semantic reconciliation engine. Ares is available as a web service and it exposes the following functionalities:
  - **applyFwdTransformation**, given as input an instance document, and the URI of the schema, which the instance is in accordance to, it transforms the instance into the ontology format, and returns the result. The ontology is the one that has been considered for building the reconciliation rules.
  - **applyBwdTransformation**, given as input data expressed in the ontology format, and the URI of the schema towards which we want to transform such data, it applies this transformation and returns the result.
  - **applyReconciliation**, given as input an instance document, and the URI of the schema of the document and the URI of the other schema involved in the reconciliation, it returns the passed instance in the format of the second schema.
Each of these components refers to a proprietary repository. Such repositories are accessible in two different ways:

- In a **tightly coupled** way; this is the way in which the proprietary component accesses it.
- In a **loosely coupled** way; this is the way in which the repository is accessible by other components. For this purpose the repositories expose their functionalities as web services.

A brief description of the repositories and their exposed services is given. A more detailed of them can be found in the respective deliverables. In particular we have:

- **Ontology Repository**, in which the domain ontologies are stored. It is managed by Athos and accessed by A* and Argos. The exposed services are:
  - `getOntologyList`: no input is required. It returns the list of the IDs of the stored ontologies;
  - `getOntology`: given the ID of an ontology it returns the ontology.

- **Annotation Repository**, is the repository of the semantic annotations: it is managed by A* and it is additionally accessed by Argos. The exposed services are:
  - `getAnnotations`, given as input an ontology ID and a document URI, it returns all the annotations for that document by means of the specified ontology;
  - `getAnnotation`, given as input an ontology ID, a document URI, and a path ID in the document, it returns the corresponding annotation.

- **Reconciliation Rules Repository**, stores the reconciliation rules and it is managed by Argos. It is additionally accessed by Ares. The exposed services are:
  - `getFwdRules`, given an ontology ID and a document URI, it returns the set of rules that allow the transformation of instances of the specified document into the ontology format.
  - `GetBwdRules`, given an ontology ID and a document URI, it returns the set of rules that allow the transformation from data expressed in the ontology format, into instances of the specified document.

- **Log Repository**, stores the log generated during the actual reconciliation performed by Ares. It can also be accessed by Argos for providing a feedback on the applied rules previously defined.

- **Themis Repository**, stores both the RDFS document (schemas) on which annotations and reconciliation rules will be defined, and RDF document (instance) on which the actual reconciliation will be applied. It is accessed by A*, Argos and Ares. The exposed services are:
  - `setRDFModel`,
  - `getRDFModelList`,
  - `getRDFModelByURI`,
  - `getRDFModelFromService`,

In the Figure 5, two other small components are represented: **OntoViz** and **ResViz**. They are two services that, given an ontology and a RDFS document, respectively, return a graphical representation of them. The functionality of these two services are used by A* and Argos for visualizing the ontology and the document schema on which the annotation and the reconciliation rules will be defined. A more detailed description of these services will be given in 6.3.2.3 and 6.3.3.

6.2 A* Architectural schema

A* has been conceived as a web application. Its architecture has been designed according to a three-tier organization (Figure 6):

- the Client-tier represents the layer where the A* clients run.
- the Application tier (A* server) where the ontology management logic is implemented.
- a back-end (Database tier) where concepts are stored and managed.
6.3 Main components of A*

6.3.1 The A* GUI

The A* Graphical User Interface is based on Web technologies. It is a light-weight client application usable through a Web browser (in its first version only through Internet Explorer). The main window of A* is organized into three main panels (Figure 7):

- **Ontology panel**, located on the left side, it shows the ontology that the user is using for the annotation. The ontology can be visualized in three different views, namely, Classes, Properties and Property path, depending on the annotation level and the subject of the annotation.

- **Resource panel**, located on the right upper side, it shows the resource document that is being annotated. The document is shown in a UML-like view. Both, the Ontology panel and the resource panel allow visualized elements to be selected, helping the user in defining the annotations without typing any character.

- **Annotation panel**, located on the right lower side, it represents the working area where annotations are created and displayed. As previously said, the user doesn’t need to type any character. Annotations are essentially built by selecting elements from the Ontology panel and the Resource panel. The appearance and the functionalities of the Annotation panel depend on the annotation level that is being performed.
Figure 7 – A* GUI: the main window

More details on the User interface and how to work with A* can be found in the Appendix F, A* User manual.

6.3.2 The Application tier

The application tier is represented by the A* server and provides all the main system functions. It is composed by the following modules: the Client Manager, the A* Application Logic, the Multilevel Automatic Support, the Document Loader, the Ontology Loader and Ontoviz.

6.3.2.1 The Client Manager

The Client Manager (Figure 8) is the sub-module that manages the communication between the Application Logic and the A* clients. The Client Manager module is in charge to:

- accept the requests from a client and return the result to the client itself. These tasks are performed by the A* gateway;
- manage the user sessions maintaining the information about the currently connected users and recognizing them at each interaction. This task is performed by the Sessions Manager;
- redirect the requests coming from the clients to the Application Logic. The response from the Application Logic is sent to the GUI Generator, which is the sub-module in charge to produce the dynamic HTML pages. Depending on the request, a specific pages, containing the requested data, is dynamically generated.
6.3.2.2 The Automatic Annotation Support

This module is in charge of providing an automatic support to the construction of the Semantic Annotation structures. Since the Annotation is a time consuming activity, the possibility to have an automatic support, aims at facilitating and easing this task. The automatic support provides a proposal of annotation, letting the user the possibility to modify such a proposal.

Currently, the Automatic Annotation Support is mainly focused on the Terminological Annotation (first level of annotation), on the Path Annotation (second level of annotation), and on the Full Annotation (fourth level of annotation). In particular we have:

- An automatic support for the Terminological level of annotation. This functionality is based on string matching algorithms. Currently a first approach has been defined and implemented.
- An automatic support for the Path based annotation. This functionality is based on a path finder algorithms. Currently a first approach is being defined.
- An automatic support for the Full Annotation. This functionality provides a translation from the Path annotation expressed in a proprietary format into OWL. Currently a first approach has been defined and implemented. (Appendix D1, FSA Example).

6.3.2.3 Document Loader and Resource Graphical Visualizer (ResViz)

The document to be annotated (document schema or resource schema) is shown in an UML-like view in the Resource panel of the GUI. There are two modules of A* that contribute to generate such pages (Figure 9):

- **DocLoader**, which is in charge to get the document to be annotated from the Themis Repository;
- **ResViz**, which actually generates the UML Class Diagram view of the document to be annotated.
6.3.2.4 DocLoader

This module is in charge of loading the schema document to be annotated. In order to allow to different kinds (formats) of resources to be annotated, the docLoader is conceived for hosting different sub-modules, each of them able to manage a different resource format and different repositories. In the following, it will described in detail the interaction with Themis repository for loading RDFS documents.

The docLoader module is called by the A* Application Logic when a certain request from the client comes. Essentially at the creation of a new job, or at the loading of an existing one. We have two cases:

- **creation of a new job**: in this case, the client first asks for the list of the available documents, and then chooses one of them from the list. To satisfy this requests, the A* Application Logic:
  - firstly, invokes the ResLoader asking to it to access the Themis Repository and retrieve the list of the available documents. The Themis method invoked by ResLoader is getRDFModelList. For each document, a set of information is retrieved, such as a name, a description and the URI. In Figure 9, this step is represented with the label 1;
  - secondly, invokes ResLoader asking to access Themis and retrieve the specific document (the one selected by the client). The Themis method invoked by ResLoader is getRDFModelByURI passing as input the URI (identifier) of the specific document. In Figure 9, this step is represented with the label 2.1.

- **loading of an existing job**: in this case, the ResLoader is not required to get the list of the documents from Themis. The A* Application Logic invokes the ResLoader just once, asking to access Themis and retrieve the needed document. As in the previous case, the Themis method invoked by ResLoader is getRDFModelByURI passing as input the URI (identifier) of the needed document. In Figure 9 this step is represented with the label 2.1.

Once the A* Application Logic has received the document, it can invoke the ResViz service passing the document to it. In this case, the specific called method is getSVGByDoc.

6.3.2.5 Resource Graphical Visualizer (ResViz)

ResViz is the service that, received a schema document, currently an RDFS document, returns an SVG document that, rendered by an SVG (Scalable Vector Graphics) visualizer (usually an Internet browser plug-in), shows the original document in a UML-like view. Such a visualization is interactive, in the sense that the visualized boxes and associations can be moved and selected (Appendix A2, Methods exposed by the SVG Resource visualization).

For generating the SVG, ResViz transforms the received document in a proprietary set of triples and, from this internal representation the SVG is generated (Appendix A1, Internal representation of resource document for generating the SVG visualization). Such internal representation is conceived to be an abstract representation of the document, independent of the original format of the document itself. In this way, ResViz is exposed as a web service accessible through the SOAP protocol.

The service exposes two operations (or methods) that allow the ResViz module to be invoked:

- **String getSVGByDoc(String doc)**. A string representing the document is passed to the service. It implies that the document is directly available to the service client (in our case, A*). This is the way in which A* currently works.

- **String getSVGByURI(String docURI)**. In this case the input parameter is represented by the URI of the document. The ResViz is delegated to actually access the document.

The WSDL document describing the system interface of the service is reported in the Appendix A3, ResViz web service Specifications.
### 6.3.3 OntoLoader and Ontology Graphical Visualizer (OntoViz)

As previously said in section 6.3.1, the ontology is in a tree-like view in the GUI where it is hosted in the left panel as HTML pages. In particular, there are two modules of A* that contribute to generate such pages (Figure 9):

- **OntoLoader**, which is in charge to get the ontology from the Ontology Repository, previously fed through Athos, and to import the ontology in an OWL format to be managed by A*;
- **OntoViz**, which actually generates the tree-like view of the ontology to be shown in the GUI.

#### 6.3.3.1 OntoLoader

This module is called by the A* Application Logic when a certain request from the client arrives. Essentially at the creation of a new job, or at the loading of an existing one. We have two cases:

- **creation of a new job**: in this case, the client first asks for the list of the available ontologies, and then choose one of them from the list. To satisfy these requests, the A* Application Logic firstly, invokes the OntoLoader asking to access the Ontology Repository and retrieve the list of the available ontologies. The Ontology Repository method invoked by OntoLoader is getOntologyList. For each document, a set of information (ontoInfo) is retrieved, such as a name, a description and the Id (identifier); In Figure 9 this step is represented with the label 3;
  
  - secondly, invokes OntoLoader asking to access the Ontology Repository and retrieve the requested ontology (the one selected by the client). The Ontology Repository method invoked by OntoLoader is getOntology, passing the ontology Id of the specific ontology as input. In Figure 9 this step is represented with the label 4.1;

- **loading of an existing job**: in this case, the OntoLoader is not required to get the list of the ontologies from the Ontology Repository. The A* Application Logic invokes the OntoLoader just once, asking to access the Ontology Repository and retrieve the needed ontology. As in the previous case, the Ontology Repository method invoked by OntoLoader is getOntology, passing the ontology Id of the specific ontology as input. In Figure 9 this step is represented with the label 4.1;

Before returning the ontology to the A* Application Logic, OntoLoader instantiates new data structures for hosting the loaded ontology. Essentially these data structures will allow to represent the...
ontology in OWL and this will be the way in which the ontology will be managed within A*. Details regarding this OWL import are reported in Appendix B1, Internal representation of the ontology for generating the HTML visualization.

6.3.3.2 Ontology Graphical Visualizer (OntoViz)

The OntoViz module returns an HTML page that shows the ontology in a tree-like view. Such a visualization is interactive, in the sense that, the nodes of the tree can be selected (Appendix B2, Methods exposed by the HTML ontology visualization).

OntoViz can be invoked internally by A* (namely by the A* Application Logic). In this case the invoked method is getOntoTreeView. Such a method, receives in input, an ontology returned by the OntoLoader and returns the HTML page showing the ontology in a tree view. In Figure 9 this step is represented with the label 4.2;

OntoViz is also exposed as a web service to be invoked by an external application (i.e., Argos). As a web service, OntoViz exposes only one operation:

- *String getOntoTreeView(String ontoId)*. A string representing the ontology Id is passed to the service. The HTML that visualizes the ontology in a tree-like view is returned as a string.

More details can be found in the Appendix B3, OntoViz web service specification.

6.3.4 A* Application Logic

The A* Application Logic is in charge to:

- Coordinate the activities of the other components of the A* server tier, as described in the previous sections
- Communicate with the Annotation Repository.

6.4 Annotation repository and the Annotation repository service

The defined annotations are stored in the Annotation repository. The Annotation Repository is made up of several components that will be described. The Annotation Repository is managed by the Annotation Repository service that can be accessed in a tightly coupled way (through procedures call), and this is the way it is used by A*, or in a loosely coupled way (as web service, the Annotation Repository service), and this is the way it is used by external applications like Argos.

6.4.1 The Annotation Repository

Annotations are organized in Jobs. A Job is identified by a document to be annotated and an ontology. The structure of the Annotation Repository takes into account such an organization. For this reason the repository is decomposed into three main parts: Jobs, Annotation connectives and Annotation Expressions.

The links among the different components of the Annotation Repository are shown in Figure 10.
6.4.1.1 The Job structure

The Job structure stores information identifying the saved Jobs. The more relevant information are the ontology Id and the document URI (resource Id).

The following table shows the structure of the Job structure repository.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JobID</td>
<td>String</td>
<td>The Job identifier</td>
</tr>
<tr>
<td>MasterAuthor</td>
<td>String</td>
<td>The creator/responsible of a job (username)</td>
</tr>
<tr>
<td>OntologyID</td>
<td>String</td>
<td>The identifier (URI) of the ontology used in this job</td>
</tr>
<tr>
<td>ResourceID</td>
<td>String</td>
<td>The identifier (URI) of the annotated resource</td>
</tr>
<tr>
<td>AuthorizedUsers</td>
<td>List</td>
<td>Users who can access/modify the Job</td>
</tr>
</tbody>
</table>

6.4.1.2 The Multilevel Annotation connectives

The following table shows the structure of the Multilevel annotation repository.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnnotID</td>
<td>String</td>
<td>The ID of the Annotation</td>
</tr>
<tr>
<td>JobID</td>
<td>String</td>
<td>The link to the Job the annotation belongs to</td>
</tr>
<tr>
<td>DS_elem</td>
<td>List</td>
<td>The ID of the annotated element. An ordered list of element IDs. If the annotation is a TSA, the set does not need to be ordered</td>
</tr>
<tr>
<td>ExprAnnotID</td>
<td>String</td>
<td>The ID of the annotation expression. This ID points to an entry of the Annotation Expression Repository</td>
</tr>
<tr>
<td>Connective</td>
<td>Enum(“=”, “&gt;”, “&lt;”, “...”)</td>
<td>Identifies the relationship between the annotated element and the annotation expression</td>
</tr>
<tr>
<td>Reliability</td>
<td>Boolean</td>
<td>Says if the annotation is still in line with the resource</td>
</tr>
<tr>
<td>CreationDate</td>
<td>Date</td>
<td>When the annotation has been performed</td>
</tr>
<tr>
<td>Author</td>
<td>String</td>
<td>Who has performed the annotation (one of the authorized users)</td>
</tr>
</tbody>
</table>
6.4.1.3 Semantic Annotation expression (SAX) structure

The following table shows the structure of the semantic annotation expression repository

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExprAnnotID</td>
<td>String</td>
<td>The annotation expression identifier</td>
</tr>
<tr>
<td>Level</td>
<td>enum(&quot;TSA&quot;, &quot;PSA&quot;, &quot;SSA&quot;, &quot;FSA&quot;)</td>
<td>Identifies the level of the annotation of the expression</td>
</tr>
<tr>
<td>Expression</td>
<td>String</td>
<td>The actual annotation expression.</td>
</tr>
<tr>
<td>CreationDate</td>
<td>Date</td>
<td>When the annotation expression has been created</td>
</tr>
<tr>
<td>Author</td>
<td>String</td>
<td>Who has defined the annotation expression (one of the authorized users)</td>
</tr>
</tbody>
</table>

6.4.2 The Annotation Repository service

The operations exposed via web service are:
- *getAnnotations*, given as input an ontology Id and a document URI, it returns all the annotations for that document with by means of the specified ontology
- *getAnnotation*: given as input an ontology Id, a document URI, and a path Id in the document, it returns the corresponding annotation.

See Appendix E2, Annotation Repository web service specification for more details.
7 Conclusions

This deliverable reported the main results of the WP4 of the A3 Project that are about the definition of a methodology for Ontology-based Semantic Annotation and the functional and architectural design of the A* tool that implements such a methodology.

Based on the SoA on annotation tools previously addressed in the WDA3.5 - State of the Art on Ontology-based Semantic Annotation methods and tools, the document firstly reports a set of requirements for the A* tool and consequently for the semantic annotation methodology behind the tool that has been identified. Such requirements mainly regard the type of annotated resources, the types of users the tool is for, the annotation formalism, the restriction on the allowed terminology, the positioning of the annotation, and the automatic support that the tool should provide.

After that, the document reports about the defined semantic annotation methodology. Such a methodology is a stepwise approach that should allow to annotate digital resources (i.e., BP Models, document schemas) in an incremental way. Four levels of annotation have been defined:

- Terminological Annotation,
- Path Annotation,
- Simple Annotation,
- Full Annotation

In the last part, the main characteristics of system design of the A* Semantic Annotation tool have been described. Some technical details and the A* user manual are reported in separated annexes.

Future works should mainly regard:

- the possibility to annotate additional kinds of documents. At the moment, mainly RDFS documents are managed by the A* tool;
- improvements of the graphical user interface and in particular of the browsing and navigability of the ontology and the resource to be annotated;
- the automatic support provided by A* for easing the user in defining the annotation;
- a more integration with the Argos tool, for improving the automatic generation of the reconciliation rules re-using the annotation.
8 References


<table>
<thead>
<tr>
<th>Programme</th>
<th>Integrating and Strengthening the European Research</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategic Objective</td>
<td>Networked businesses and governments</td>
</tr>
<tr>
<td>Integrated Project / Programme Title</td>
<td>Advanced Technologies for Interoperability of Heterogeneous Enterprise Networks and their Application</td>
</tr>
<tr>
<td>Acronym</td>
<td>ATHENA</td>
</tr>
<tr>
<td>Project No</td>
<td>507849</td>
</tr>
<tr>
<td>ATHENA – Project Name</td>
<td>Knowledge Support and Semantic Mediation Solutions</td>
</tr>
<tr>
<td>ATHENA A - Project No</td>
<td>A3</td>
</tr>
</tbody>
</table>

**Deliverable Nr: D.A3.3**

Semantic Annotation language and tool for Information and Business Processes

Appendix A: ResViz

**Work Package – A3**

Leading Partner: CNR-IASI

Security Classification: Project Participants (PP)

February 14, 2006

Version 1.0
Table of contents

APPENDIX A: RESVIZ  
A. 1 INTERNAL REPRESENTATION OF RESOURCE DOCUMENT FOR GENERATING THE SVG VISUALIZATION 1  
A. 2 METHODS EXPOSED BY THE SVG RESOURCE VISUALIZATION 3  
A. 3 RESVIZ WEB SERVICE SPECIFICATION 7

Table of Figures

Figure 1 – A path represented by a single Class ................................................................. 4  
Figure 2 - A Property whose range is a Class as last element of the path ............................. 4  
Figure 3 – A Property whose range is a basic type as last element of the path .................... 4  
Figure 4 – An enumeration type as last element of the path .............................................. 5  
Figure 5 – An enumeration item as last element of the path .............................................. 5
Appendix A: ResViz

A. 1 Internal representation of resource document for generating the SVG visualization

The objective of the ResViz module is to transform the resource document into an SVG document that can be visualized in the GUI.

This process is organized into two steps:
1. transformation of the resource document into an intermediate internal representation;
2. transformation of the internal representation into SVG.

The aim of the internal representation is to generate the SVG independently of the format of the resource document. This implies that for dealing with a new format it will be enough to define the transformation involved at step 1 between the new format and the intermediate representation. The step 2 will always the same whatever is the format of the resource document.

Since the SVG visualization intends to show the resource document into an UML Class Diagram like view, the intermediate representation needs to describe Classes, Associations and Attributes. That is done in the following way:

- **Classes** are represented through a list of pairs of the following format:
  
  \[(class\_Id, class\_Label)\]

  where \(class\_Id\) and \(class\_Label\) correspond respectively to the identifier and the label of the Class.

- **Associations** are represented through a list of triples of the following format:
  
  \[((classS\_Id, classS\_Label),(assoc\_Id, assoc\_Label, 'Association'),(classD\_Id, classD\_Label))\]

  where \(classS\) and \(classD\) represent respectively the source and the destination classes of the association and \(assoc\) represents the association linking the two classes;

- **Attributes** are represented through a list of triples of the following format:
  
  \[(class\_Id, class\_Label),(attr\_Id, attr\_Label, 'Attribute'),XMLtype)\]

  where the Class \(class\) has an Attribute \(attr\) whose type is \(XMLType\). \(XMLType\) is one of the XMLSchema Datatypes (i.e., int, string, ...).

In the following, the criteria for transforming RDF documents into the intermediate representation is shown:

- If in the RDF document there is an \texttt{rdf:Class}, a pair representing a Class is added to list of pairs of the resource document

- If in the RDF document there is an \texttt{rdf:property} with
  - ID=\texttt{prop\_Id} and label=\texttt{prop\_Label},
  - and linking two Classes
  - one (the domain Class) with ID=\texttt{classD\_Id} and label=\texttt{classD\_Label}
  - the other one (the range Class) with ID=\texttt{classR\_Id} and label=\texttt{classR\_Label}
  - then the following triple is created:
    - \[((classD\_Id, classD\_Label),( prop\_Id, prop\_Label, 'Association'),( classR\_Id, classR\_Label))\]

- If in the RDF document there is an \texttt{rdf:property} with
  - ID=\texttt{prop\_Id} and label=\texttt{prop\_Label},
and linking an RDF class with an basic type (XMLdatatype)
- Class (domain class) with ID=classD_Id and label= classD_Label
- XMLDatatype (range) whose type is XMLType (i.e. int, string,…)
- then the following triple is created:
  - ((classD_Id, classD_Label),( prop_Id, prop_Label, 'Attribute'),( XMLType, ''))
  - 
- If the RDF document there is an rdf:property with
  - ID=prop_Id and label=prop_Label,
  - and linking an RDF classes with an enumerated datarange
  - Class (domain class) with ID=classD_Id and label= classD_Label
  - DataRange (range) composed by a list of values ([value1,value2,…]) instances of a specific XMLType
  - then the following triples are created:
  - a triple to describe the datarange
    - ((classD_Id, classD_Label),( prop_Id, prop_Label, 'Association'),( prop_Id+'<<enumeration>>',''))
  - a triple for each enumerated value appeared into the datarange
    - ((prop_Id+'<<enumeration>>', ''),( value,'', 'Attribute'),( XMLType, ''))
A. 2 Methods exposed by the SVG Resource visualization

The ResViz module allows to visualize in a UML Class Diagram like view the resource document that has to be annotated. Furthermore, since the user has to interact with this diagram in order to select single elements or paths that he/she intends to annotate, a set of JavaScript methods are exposed by this visualization. Such methods listed here are described in the following:

- `getPath`, returns the selected path
- `setPath`, selects a given path
- `clearPath`, deselect the current selected path.

A2.1 getPath method

- Selecting a path in the Resource panel, the `getPath` method will return the sequence of Classes and Properties the path itself is composed by. The returned path will be represented by the concatenation of the IDs of each element (Class or Property), separated by a dot.

Method signature: the `signature` of the JavaScript method is `String getPath();`

Input: the method does not require any input parameter.

Output: the method output is a String representing the selected path.

- The structure of the returned String will depend on the elements of the path that has been selected. The following cases are possible:
  1. the path is represented by a single `Class`,
  2. the last element of the selected path is a `Property` whose range is a `Class`,
  3. the last element of the selected path is a `Property` whose range is a `basic type`,
  4. the last element of the selected path is a `Property` whose range is an `enumeration`,
  5. the last element of the selected path is an `item` of an `enumeration`. 
1. **The path is represented by a single Class**
The returned string is the ID of the selected class.
An example is given in Figure 1. The label of boxes (Classes) and arcs (Properties) represents the ID:

![Figure 1 - A path represented by a single Class](image)

The returned String: *Order*.

2. **The last element of the selected path is a Property whose range is a Class**
The returned string is the concatenation of IDs of Classes and Properties, including also the ID of the Class that represents the range of the final Property.
An example is given in Figure 2.

![Figure 2 - A Property whose range is a Class as last element of the path](image)


3. **The last element of the selected path is a Property whose range is a basic type**
The returned string is the concatenation of IDs of Classes and Properties ends with a semicolon followed by the basic type of the Property.
An example is given in Figure 3.

![Figure 3 - A Property whose range is a basic type as last element of the path](image)

The returned String: *Order.has_BuyerParty.BuyerParty.BuyerParty_has_Party.Party.has_Contact.Contact:Contact_has_Name:string*.

4. **The last element of the selected path is a Property whose range is an enumeration**
The returned string is the concatenation of IDs of Classes and Properties that ends with curly brackets in which there is the list of enumeration items separated by a comma. An example is given in Figure 4.

![Figure 4 – An enumeration type as last element of the path](image)

The returned String: `Order.hasTransportationTerms{EXW, FRC}

5. The last element of the selected path is an item of an enumeration

The returned string is the concatenation of IDs of Classes and Properties that ends with round brackets in which there is the selected enumeration item. An example is given in Figure 5.

![Figure 5 – An enumeration item as last element of the path](image)

The returned String: `Order.has_TransportationTerms(EXW)`

### A2.2 setPath method

The method setPath allows to highlight a path in the resource panel. **Method signature**: the signature of the javascript method is `void setPath(String path, String mode)`

- **Input**:  
  - `path` parameter, a string value representing the path to be selected (formatted as seen in section )  
  - `mode` parameter to distinguish between the current annotation and the ones already stored in the db. Allowed values : “current” and “done”. This parameter is currently used to give a different color to current and already annotated paths.  
- **Output**: the method does not return any value.
A.2.3 clearPath method

The method `clearPath` deselect the path on which we are currently working (mode='current').

**Method signature:** the **Signature** of the javascript method is `void clearPath()`.
A. 3 ResViz web service specification

Below, the WSDL document of the ResViz web service is reported.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<wsdl:message name="getSVGByDocResponse">
  <wsdl:part name="getSVGByDocReturn" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="getSVGByDocRequest">
  <wsdl:part name="in0" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="getSVGByURIResponse">
  <wsdl:part name="getSVGByURIReturn" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="getSVGByURIRequest">
  <wsdl:part name="in0" type="xsd:string"/>
</wsdl:message>

<wsdl:portType name="ResViz">
  <wsdl:operation name="getSVGByURI" parameterOrder="in0">
    <wsdl:input message="intf:getSVGByURIRequest" name="getSVGByURIRequest"/>
    <wsdl:output message="intf:getSVGByURIResponse" name="getSVGByURIResponse"/>
  </wsdl:operation>

  <wsdl:operation name="getSVGByDoc" parameterOrder="in0">
    <wsdl:input message="intf:getSVGByDocRequest" name="getSVGByDocRequest"/>
    <wsdl:output message="intf:getSVGByDocResponse" name="getSVGByDocResponse"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="ResVizSoapBinding" type="intf:ResViz">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
</wsdl:binding>
</wsdl:definitions>
```
<wsdl:operation name="getSVGByURI">
  <wsdl:input name="getSVGByURIRequest">
    <wsdl:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="ResViz" use="encoded"/>
  </wsdl:input>
  <wsdl:output name="getSVGByURIResponse">
    <wsdl:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="ResViz" use="encoded"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="getSVGByDoc">
  <wsdl:input name="getSVGByDocRequest">
    <wsdl:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="ResViz" use="encoded"/>
  </wsdl:input>
  <wsdl:output name="getSVGByDocResponse">
    <wsdl:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="ResViz" use="encoded"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:service name="ResVizService">
  <wsdl:port binding="intf:ResVizSoapBinding" name="ResViz">
    <wsdl:address location="http://leks-pub.iasi.cnr.it/astar/services/ResViz"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
Programme
Integrating and Strengthening the European Research
Strategic Objective
Networked businesses and governments

Integrated Project / Programme Title
Advanced Technologies for Interoperability of Heterogeneous Enterprise Networks and their Application

Acronym
ATHENA

Project No
507849

ATHENA – Project Name
Knowledge Support and Semantic Mediation Solutions

ATHEN A - Project No
A3

Deliverable Nr: D.A3.3

Semantic Annotation language and tool for Information and Business Processes

Appendix B: OntoViz

Work Package – A3

Leading Partner: CNR-IASI

Security Classification: Project Participants (PP)

February 14, 2006

Version 1.0
Table of contents

APPENDIX B: ONTOVIZ

B1. INTERNAL REPRESENTATION OF THE ONTOLOGY FOR GENERATING THE HTML VISUALIZATION 1
B2. METHODS EXPOSED BY THE HTML ONTOLOGY VISUALIZATION 4
B3. ONTOVIZ WEB SERVICE SPECIFICATION 6
Appendix B: OntoViz

B1. Internal representation of the ontology for generating the HTML visualization

The ontology (that is a directed acyclic graph) is visualized as a set of tree.

Every concept (non basic type) in the ontology represents a root of a single tree. A tree is composed by all paths that link the root concept to a generic basic type. If a concept or basic type is related with more than one concept it will be duplicated.

The internal representation of a tree is a structure organized in the following way:

```
{conceptIdToVis: {kind: string, text: string, childsNodes: list}}
```

- `conceptIdToVis` is a field formed by a `conceptId` concatenate with an alphanumeric code that allows to distinguish between same concepts that appear in different tree (duplicated concepts)
- Field `kind` is an enumerated value that can be assume values ‘Class’ or ‘property’ used to determine the icon to apply in the visualization
- Field `text` represent the label of the concept visualized
- `childNodes` is a list of couple (`conceptIdToVis`, `conceptId`) that identify necessary information about the child of vertex in the tree represented.

For instance, the internal representation for the tree associate to `Buyer_BusinessActor` could be:

```

{ 'Buyer_BusinessActor__13': {
    'kind': 'class',
    'text': 'Buyer_BusinessActor',
    'childNodes': [
        ('has_TelephoneNumber__13_1', 'has_TelephoneNumber'),
        ('has_FaxNumber__13_2', 'has_FaxNumber'),
        ('has_emailAddress__13_3', 'has_emailAddress'),
        ('has_URI__13_4', 'has_URI'),
        ('has_PostalAddress__13_5', 'has_PostalAddress'),
        ('has_PartyTaxID__13_6', 'has_PartyTaxID'),
        ('relTo_ContactPerson__13_7', 'relTo_ContactPerson')],
    'has_TelephoneNumber__13_1': {
        'text': 'has_TelephoneNumber:STRING',
        'kind': 'property'},
    'has_FaxNumber__13_2': {

```
'text': 'has_FaxNumber:INT',
'kind': 'property'},
'has_emailAddress__13_3': {
'text': 'has_emailAddress:STRING',
'kind': 'property'},
'has_URI__13_4': {
'text': 'has_URI:STRING',
'kind': 'property'},
'has_PostalAddress__13_5': {
'kind': 'property',
'text': 'has_PostalAddress: PostalAddress',
'childNodes': [
  ('PostalAddress_ComplexAttribute__13_5_1',
   'PostalAddress_ComplexAttribute')],
'PostalAddress_ComplexAttribute__13_5_1': {
'text': 'PostalAddress_ComplexAttribute',
'childNodes': [
  ('hasPart_PostalCode__13_5_1_1', 'hasPart_PostalCode'),
  ('hasPart_PostOfficeBoxNumber__13_5_1_2', 'hasPart_PostOfficeBoxNumber'),
  ('hasPart_StreetName__13_5_1_3', 'hasPart_StreetName'),
  ('hasPart_County__13_5_1_4', 'hasPart_County'),
  ('hasPart_BuildingNumber__13_5_1_5', 'hasPart_BuildingNumber'),
  ('hasPart_CountryName__13_5_1_6', 'hasPart_CountryName'),
  ('hasPart_City__13_5_1_7', 'hasPart_City'),
  ('hasPart_CountryCode__13_5_1_8', 'hasPart_CountryCode')],
'hasPart_PostalCode__13_5_1_1': {
'text': 'hasPart_PostalCode:STRING',
'kind': 'property'},
'hasPart_PostOfficeBoxNumber__13_5_1_2': {
'text': 'hasPart_PostOfficeBoxNumber:STRING',
'kind': 'property'},
'hasPart_StreetName__13_5_1_3': {
'text': 'hasPart_StreetName:STRING',
'kind': 'property'},
'hasPart_County__13_5_1_4': {
B2. Methods exposed by the HTML ontology visualization

The OntoViz module allows to visualize in tree like view the ontology, by applying a spanning tree technique. Such a technique allows a graph to be represented as a set of trees. Due to that, given nodes in the ontology as a graph are duplicated when the spanning tree is applied. Such visualization is offered as an HTML page.

Furthermore, since the user has to interact with this visualization of the ontology in order to select paths that he/she intends to annotate, a set of JavaScript methods are exposed. Such methods listed here are described in the following:

- `getPath`, returns the selected path
- `setPath`, selects a given path
- `clearPath`, deselect the current selected path.

B.2 `getFullPath` method

Selecting a path in the Ontology panel, the `getFullPath` method will return the sequence of Classes and Properties the path itself is composed by. The returned path will be represented by the concatenation of the IDs of each element (Class or Property), separated by a dot.

Method signature: the signature of the JavaScript method is `String getFullPath();`

Input: the method does not require any input parameter.

Output: the method output is a String representing the selected path.

The structure of the returned String will depend on the elements of the path that has been selected. The following cases are the same as the ones considered for the visualization of the resource document in the SVG format:

1. the path is represented by a single Class,
2. the last element of the selected path is a Property whose range is a Class,
3. the last element of the selected path is a Property whose range is a basic type,
4. the last element of the selected path is a Property whose range is an enumeration,
5. the last element of the selected path is an item of an enumeration.

Please refer to Appendix A2 for details.
A2.2 setPath method

The method `setPath` allows to highlight a path in the ontology panel.

**Method signature:** the **signature** of the javascript method is `void setPath(String path, String mode)`

the **signature** of the JavaScript method is: `void setPath(String path, String mode)``

**Input:**
- `path` parameter, a string value representing the path to be selected (formatted as seen in section
- `mode` parameter to distinguish between the current annotation and the ones already stored in the db. Allowed values: “current” and “done”. This parameter is currently used to give a different color to current and already annotated paths.

**Output:** the method does not return any value.

A.2.3 clearPath method

The method `clearPath` deselect the path on which we are currently working (mode=‘current’)

**Method signature:** the **Signature** of the JavaScript method is `void clearPath()`
B3. OntoViz web service specification

Below, the WSDL document of the OntoViz web service is reported.

```xml
<wsdl:definitions targetNamespace="OntoViz" xmlns="http://schemas.xmlsoap.org/wsdl/

  <wsdl:message name="getOntoTreeViewRequest">
    <wsdl:part name="in0" type="xsd:string"/>
  </wsdl:message>

  <wsdl:message name="getOntoTreeViewResponse">
    <wsdl:part name="getOntoTreeViewReturn" type="xsd:string"/>
  </wsdl:message>

  <wsdl:portType name="OntoViz">
    <wsdl:operation name="getOntoTreeView" parameterOrder="in0">
      <wsdl:input message="intf:getOntoTreeViewRequest" name="getOntoTreeViewRequest"/>
      <wsdl:output message="intf:getOntoTreeViewResponse" name="getOntoTreeViewResponse"/>
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="OntoVizSoapBinding" type="intf:OntoViz">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getOntoTreeView">
      <wsdlsoap:operation soapAction=""/>
      <wsdlsoap:input body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="OntoViz" use="encoded"/>
    </wsdlsoap:input>
    <wsdlsoap:output body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="OntoViz" use="encoded"/>
  </wsdl:operation>
</wsdl:binding>
```


### Programme

Integrating and Strengthening the European Research Strategic Objective

**Networked businesses and governments**

Integrated Project / Programme Title

Advanced Technologies for Interoperability of Heterogeneous Enterprise Networks and their Application

**Acronym**

**ATHENA**

Project No

507849

ATHENA – Project Name

Knowledge Support and Semantic Mediation Solutions

ATHENA - Project No

A3

---

**Deliverable Nr: D.A3.3**

Semantic Annotation language and tool for Information and Business Processes

Appendix C: Ontology import

**Work Package – A3**

Leading Partner: CNR-IASI

Security Classification: Project Participants (PP)

February 14, 2006

Version 1.0
Table of contents

APPENDIX C: OWL ONTOLOGY IMPORT

Table of Figures

Figure 1 - Class diagram of RESOURCE package included in OntoLoader Module .................................. 1
Figure 2 - Class diagram of package GRAPH_OWL included in OntoLoader Module.................................. 2
Figure 3 - Class diagram of OWLOntology class .................................................................................. 2
Appendix C: OWL Ontology import

OntoLoader instantiates new data structures for hosting the loaded ontology. Essentially these data structures will allow the ontology to be represented in OWL, and this will be the way in which the ontology will be managed within A*.

The OntoloLoader module includes several packages:

- The RESOURCE package includes classes that represent the possible OWL constructs. As shown in Figure 1 two fundamental subClasses are defined. The other classes inherit from them. The BinaryRelation class represents the class that comprises the OWL constructs to express the relation between two entity: as a property that links a domain with respective range or the axioms that link two classes or properties. Classes that inherit by RDFResource directly derive from specialization hierarchy defined between the OWL constructs. Classes highlighted in dark grey are not significant to understand system functionality. They are Zope internal classes that the programmer has to extend to define the other classes.

Figure 1 - Class diagram of RESOURCE package included in OntoLoader Module
• The GRAPH_OWL package is composed by a single class `Graph`, it represents a graph, in which the vertexes are the classes identified by the existence of a relation (a "BinaryRelation") between them. The Graph class allows to add a relation, to obtain the inner edge and outer edge concerning a specific vertex and to found a relation with a specific “id”

![Class diagram of package GRAPH_OWL included in OntoLoader Module](image1)

• The ONTO_OWL Package is composed by a single class OWLOntology (Figure 3) that represent the container of different object elements. Every object OWLOntology has associated three instances of Graph Class, representing respectively, the property graph, the subProperty graph and the SubClass graph.

![Class diagram of OWLOntology class](image2)
An OWL ontology is represented in Zope by an OWLOntology object that functions by a container of different objects:

- Object that are instances of classes that are in specialization hierarchy with the class OWLClass
- Graph Object. An Object OWLOntology includes three Graph instances representing respectively, the property graph, the subProperty graph and the SubClass graph. The EquivalentClass has treated as two different SubClass (with argument inverted
Deliverable Nr: D.A3.3

Semantic Annotation language and tool for Information and Business Processes

Appendix D: Examples of annotation

Work Package – A3

Leading Partner: CNR-IASI

Security Classification: Project Participants (PP)

February 14, 2006

Version 1.0
Table of contents

APPENDIX D: EXAMPLES OF ANNOTATION 1

B1. AN EXAMPLE OF FULL ANNOTATION 1
   D1.1. N3 Code representing RDF(S) of the AIDIMA order simplified document 1
   D1.2. N3 Code representing OWL Ontology used to annotate the PurchaseOrder document 7

B2. FSA EXAMPLE 25
   D2.1. Translation of PSA into OWL 25
   D2.2. Translation of SSA into OWL 26
Appendix D: Examples of annotation

B1. An example of Full Annotation

In this section is presented an example of FSA.

In the first and second subsection are respectively shown a simplified version of the AIDIMA order document and a simplified version of the Reference Ontology used to build annotations. Both of them are presented into N3 format that is a compact format to express RDF or OWL documents.

At the third subsection is shown an example of FSA of a specific ds-elem of AIDIMA document using the RO proposed. The subsection E.3.1 presents the FSA associate to each path defined in PSAV while the subsection E.3.2 proposes a way to express the abstract operator introduced in SSA.

D1.1. N3 Code representing RDF(S) of the AIDIMA order simplified document

In this section is showed the n3 code of AIDIMA order simplified document.

```
#Processed by Id: cwm.py,v 1.148 2004/03/21 04:24:32 timbl Exp
#  using base file:/aidima.rdf
# Notation3 generation by
#     notation3.py,v 1.153 2004/03/21 04:24:35 timbl Exp

# Base was: file:/aidima.rdf
@prefix : <#> .
@prefix XML: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

:Order    a rdfs:Class .

:address   a rdfs:Class .

:addressDetails   a rdfs:Class .

:amount      a rdfs:Class .

:buyerInfo   a rdfs:Class .

:buyerInfo_has_organisationInfo   a rdf:Property;
   rdfs:domain :buyerInfo;
   rdfs:range :organisationInfo .

:buyerOrderNumber   a rdfs:Class .
```
:contactPerson a rdfs:Class.

:date a rdfs:Class.

:dimensions a rdfs:Class.

:figures a rdfs:Class.

:has_URL a rdf:Property;
   rdfs:domain :organisationInfo;
   rdfs:range XML:string.

:has_address a rdf:Property;
   rdfs:domain :addressDetails;
   rdfs:range :address.

:has_addressDetails a rdf:Property;
   rdfs:domain :organisationInfo;
   rdfs:range :addressDetails.

:has_amount a rdf:Property;
   rdfs:domain :figures;
   rdfs:range :amount.

:has_buyerInfo a rdf:Property;
   rdfs:domain :orderHeader;
   rdfs:range :buyerInfo.

:has_buyerOrderNumber a rdf:Property;
   rdfs:domain :orderHeader;
   rdfs:range :buyerOrderNumber.

:has_city a rdf:Property;
   rdfs:domain :address;
   rdfs:range XML:string.

:has_contactPerson a rdf:Property;
   rdfs:domain :organisationInfo;
   rdfs:range :contactPerson.

:has_country a rdf:Property;
   rdfs:domain :address;
<table>
<thead>
<tr>
<th>IP- Project</th>
<th>ATHENA</th>
<th>IP- Project - No</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATHENA - Project</td>
<td>Programme</td>
<td>ATHENA - Project Number</td>
</tr>
<tr>
<td>Document</td>
<td></td>
<td>Date</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25.02.06</td>
</tr>
</tbody>
</table>

```xml
rdfs:range XML:string .

:has_day a rdf:Property;
  rdfs:domain :date;
  rdfs:range XML:string .

:has_depth a rdf:Property;
  rdfs:domain :dimensions;
  rdfs:range XML:string .

:has_description a rdf:Property;
  rdfs:domain :productRecord;
  rdfs:range XML:string .

:has_dimensions a rdf:Property;
  rdfs:domain :productAttributes;
  rdfs:range :dimensions .

:has_eMail a rdf:Property;
  rdfs:domain :organisationInfo;
  rdfs:range XML:string .

:has_fax a rdf:Property;
  rdfs:domain :organisationInfo;
  rdfs:range XML:string .

:has_figures a rdf:Property;
  rdfs:domain :paymentInformation;
  rdfs:range :figures .

:has_grandTotal a rdf:Property;
  rdfs:domain :amount;
  rdfs:range XML:string .

:has_height a rdf:Property;
  rdfs:domain :dimensions;
  rdfs:range XML:string .

:has_hour a rdf:Property;
  rdfs:domain :date;
  rdfs:range XML:string .

:has_identifier a rdf:Property;
```
rdfs:domain :buyerOrderNumber;
  rdfs:range XML:string .

:has_minutes  a rdf:Property;
  rdfs:domain :date;
  rdfs:range XML:string .

:has_month    a rdf:Property;
  rdfs:domain :date;
  rdfs:range XML:string .

:has_name     a rdf:Property;
  rdfs:domain :contactPerson;
  rdfs:range XML:string .

:has_number   a rdf:Property;
  rdfs:domain :address;
  rdfs:range XML:string .

:has_orderDate a rdf:Property;
  rdfs:domain :orderHeader;
  rdfs:range :orderDate .

:has_orderHeader a rdf:Property;
  rdfs:domain :Order;
  rdfs:range :orderHeader .

:has_organisationName a rdf:Property;
  rdfs:domain :organisationInfo;
  rdfs:range XML:string .

:has_pOBox    a rdf:Property;
  rdfs:domain :address;
  rdfs:range XML:string .

:has_paymentInformation a rdf:Property;
  rdfs:domain :orderHeader;
  rdfs:range :paymentInformation .

:has_phone    a rdf:Property;
  rdfs:domain :organisationInfo;
  rdfs:range XML:string .
:has_postalCode a rdf:Property;
    rdfs:domain :address;
    rdfs:range XML:string .

:has_productAttributes a rdf:Property;
    rdfs:domain :productRecord;
    rdfs:range :productAttributes .

:has_productCode a rdf:Property;
    rdfs:domain :productRecord;
    rdfs:range XML:string .

:has_productRecord a rdf:Property;
    rdfs:domain :productsInfo;
    rdfs:range :productRecord .

:has_productUnitaryCost a rdf:Property;
    rdfs:domain :productRecord;
    rdfs:range XML:string .

:has_productsInfo a rdf:Property;
    rdfs:domain :Order;
    rdfs:range :productsInfo .

:has_province a rdf:Property;
    rdfs:domain :address;
    rdfs:range XML:string .

:has_quantity a rdf:Property;
    rdfs:domain :productRecord;
    rdfs:range XML:string .

:has_requestedDeliveryDate a rdf:Property;
    rdfs:domain :orderHeader;
    rdfs:range :requestedDeliveryDate .

:has_street a rdf:Property;
    rdfs:domain :address;
    rdfs:range XML:string .

:has_supplierInfo a rdf:Property;
    rdfs:domain :orderHeader;
    rdfs:range :supplierInfo .
:has_taxNumber a rdf:Property;
  rdfs:domain :organisationInfo;
  rdfs:range XML:string .

:has_transportationTerms a rdf:Property;
  rdfs:domain :orderHeader;
  rdfs:range XML:string .

:has_width a rdf:Property;
  rdfs:domain :dimensions;
  rdfs:range XML:string .

:has_year a rdf:Property;
  rdfs:domain :date;
  rdfs:range XML:string .

:orderDate a rdfs:Class .

:orderDate_has_date a rdf:Property;
  rdfs:domain :orderDate;
  rdfs:range :date .

:orderHeader a rdfs:Class .

:organisationInfo a rdfs:Class .

:organisationName a rdfs:Class .

:paymentInformation a rdfs:Class .

:productAttributes a rdfs:Class .

:productRecord a rdfs:Class .

:productsInfo a rdfs:Class .

:requestedDeliveryDate a rdfs:Class .

:requestedDeliveryDate_has_date a rdf:Property;
  rdfs:domain :requestedDeliveryDate;
  rdfs:range :date .
D1.2. N3 Code representing OWL Ontology used to annotate the PurchaseOrder document

In this section is showed the n3 code of Reference Ontology.

```n3
@prefix : <http://www.w3.org/2002/07/owl#> .
@prefix RO: <#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

RO:ATTRIBUTES     a :ObjectProperty .

RO:BOOLEAN     a :Class;
    rdfs:subClassOf [ a :Restriction;
        :cardinality "\n\t\t1\n\t\t"^^xsd:nonNegativeInteger;

RO:BUSINESS_ACTOR     a :Class .

RO:BUSINESS_OBJECT     a :Class .

RO:BUSINESS_OBJECT_DOCUMENT     a :Class;
    rdfs:subClassOf RO:BUSINESS_OBJECT .

RO:BusinessPartner_BusinessActor     a :Class;
    rdfs:subClassOf RO:BUSINESS_ACTOR,
    [ a :Restriction;
        :minCardinality "\n\t0\n\t"^^xsd:nonNegativeInteger;
    ] .

RO:supplierInfo     a rdfs:Class .

:supplierInfo_has_organisationInfo     a rdf:Property;
    rdfs:domain :supplierInfo;
    rdfs:range :organisationInfo .

#ENDS
```
:onProperty RO:relTo_ContactPerson ],
    [ a :Restriction;
  :minCardinality "\n\t1\n\t"^^xsd:nonNegativeInteger;
  :onProperty RO:has_PostalAddress ],
    [ a :Restriction;
  :allValuesFrom RO:STRING;
  :onProperty RO:has_TelephoneNumber ],
    [ a :Restriction;
  :minCardinality "\n\t0\n\t"^^xsd:nonNegativeInteger;
  :onProperty RO:has_TelephoneNumber ],
    [ a :Restriction;
  :allValuesFrom RO:STRING;
  :onProperty RO:has_emailAddress ],
    [ a :Restriction;
  :minCardinality "\n\t0\n\t"^^xsd:nonNegativeInteger;
  :onProperty RO:has_emailAddress ],
    [ a :Restriction;
  :allValuesFrom RO:INT;
  :onProperty RO:has_FaxNumber ],
    [ a :Restriction;
  :minCardinality "\n\t0\n\t"^^xsd:nonNegativeInteger;
  :onProperty RO:has_FaxNumber ],
    [ a :Restriction;
  :allValuesFrom RO:STRING;
  :onProperty RO:has_URI ],
    [ a :Restriction;
  :minCardinality "\n\t1\n\t"^^xsd:nonNegativeInteger;
  :onProperty RO:has_URI ],
    [ a :Restriction;
  :allValuesFrom RO:STRING;
  :onProperty RO:has_PartyTaxID ],
    [ a :Restriction;
:cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:has_PartyTaxID }  .

RO:Buyer_BusinessActor    a :Class;
    rdfs:label "PU",
    "purchaser",
    "purchasing unit";
    rdfs:subClassOf RO:Organization_BusinessActor .

RO:COMPLEX_ATTRIBUTE    a :Class .

RO:ContactPerson_BusinessActor    a :Class;
    rdfs:subClassOf RO:BUSINESS_ACTOR,
    [   a :Restriction;
        :allValuesFrom RO:STRING;
        :onProperty RO:has_PersonSurname ],
    [   a :Restriction;
        :cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
        :onProperty RO:has_PersonSurname ],
    [   a :Restriction;
        :allValuesFrom RO:STRING;
        :onProperty RO:has_PersonFirstName ],
    [   a :Restriction;
        :cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
        :onProperty RO:has_PersonFirstName ]  .

RO:CostAndFreight    a :Thing .

RO:DECOMPOSITION    a :ObjectProperty .

RO:DateTime_ComplexAttribute    a :Class;
    rdfs:label "date",
    "time";
    rdfs:subClassOf RO:COMPLEX_ATTRIBUTE,
    [   a :Restriction;
        :allValuesFrom RO:INT;
        :onProperty RO:hasPart_Hour ],
    [ }
a :Restriction;
:cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:hasPart_Hour ],
  [ a :Restriction;
:allValuesFrom RO:INT;
:onProperty RO:hasPart_Minute ],
  [ a :Restriction;
:cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:hasPart_Minute ],
  [ a :Restriction;
:allValuesFrom RO:INT;
:onProperty RO:hasPart_Day ],
  [ a :Restriction;
:cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:hasPart_Day ],
  [ a :Restriction;
:allValuesFrom RO:INT;
:onProperty RO:hasPart_Month ],
  [ a :Restriction;
:cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:hasPart_Month ],
  [ a :Restriction;
:allValuesFrom RO:INT;
:onProperty RO:hasPart_Year ],
  [ a :Restriction;
:cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:hasPart_Year ],
  [ a :Restriction;
:allValuesFrom RO:INT;
:onProperty RO:hasPart_Second ],
  [ a :Restriction;
:cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:hasPart_Second ] .
RO:DeliveryDate_ComplexAttribute    a :Class;
   rdfs:subClassOf RO:DateTime_ComplexAttribute .

RO:ExWorks      a :Thing .

RO:FLOAT        a :Class;
   rdfs:subClassOf  
     [ a :Restriction;
       :cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;

RO:FreeCarrier   a :Thing .

RO:FreeOnBoard   a :Thing .

RO:INT          a :Class;
   rdfs:subClassOf  
     [ a :Restriction;
       :cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;

RO:IssueDate_ComplexAttribute   a :Class;
   rdfs:subClassOf RO:DateTime_ComplexAttribute .

RO:IssuingPurchaseOrder_Process  a :Class;
   rdfs:label "IssuingPO",
     "SendingPurchaseOrder";
   rdfs:subClassOf RO:PROCESS .

RO:OrderLine_BusinessObjectDocument  a :Class;
   rdfs:subClassOf RO:BUSINESS_OBJECT_DOCUMENT,
     [ a :Restriction;
       :cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
       :onProperty RO:relTo_Product ],
     [ a :Restriction;
       :allValuesFrom RO:INT;
       :onProperty RO:has_Quantity ],
     [ a :Restriction;
       :cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:has_Quantity },
    [a :Restriction;
     :allValuesFrom RO:FLOAT;
     :onProperty RO:has_LinePrice ],
    [
      a :Restriction;
      :cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
      :onProperty RO:has_LinePrice ] .

RO:Organization_BusinessActor a :Class;
    rdfs:subClassOf RO:BusinessPartner_BusinessActor .

RO:PROCESS a :Class .

RO:PostalAddress_ComplexAttribute a :Class;
    rdfs:subClassOf RO:COMPLEX_ATTRIBUTE,
    [a :Restriction;
     :allValuesFrom RO:STRING;
     :onProperty RO:hasPart_BuildingNumber ],
    [a :Restriction;
     :minCardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
     :onProperty RO:hasPart_BuildingNumber ],
    [a :Restriction;
     :maxCardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
     :onProperty RO:hasPart_BuildingNumber ],
    [a :Restriction;
     :allValuesFrom RO:STRING;
     :onProperty RO:hasPart_PostalCode ],
    [a :Restriction;
     :cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
     :onProperty RO:hasPart_PostalCode ],
    [a :Restriction;
     :allValuesFrom RO:STRING;
     :onProperty RO:hasPart_StreetName ],
    [a :Restriction;
<table>
<thead>
<tr>
<th>IP- Project</th>
<th>ATHENA</th>
<th>IP- Project - No</th>
<th>507849</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATHENA - Project</td>
<td>Programme</td>
<td>ATHENA - Project Number</td>
<td></td>
</tr>
<tr>
<td>Document</td>
<td></td>
<td>Date</td>
<td>25.02.06</td>
</tr>
</tbody>
</table>

```xml
:minCardinality "\n\t0\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:hasPart_StreetName },

[  
a :Restriction;
:maxCardinality "\n\t1\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:hasPart_StreetName ],

[  
a :Restriction;
:allValuesFrom RO:STRING;
:onProperty RO:hasPart_CountryCode },

[  
a :Restriction;
:allValuesFrom RO:STRING;
:onProperty RO:hasPart_CountryName },

[  
a :Restriction;
:minCardinality "\n\t0\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:hasPart_CountryName },

[  
a :Restriction;
:maxCardinality "\n\t1\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:hasPart_CountryName ],

[  
a :Restriction;
:allValuesFrom RO:STRING;
:onProperty RO:hasPart_City },

[  
a :Restriction;
:allValuesFrom RO:STRING;
:onProperty RO:hasPart_PostOfficeBoxNumber ],

[  
a :Restriction;
:cardinality "\n\t1\n\t\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:hasPart_PostOfficeBoxNumber ],
```
a :Restriction;
:allValuesFrom RO:STRING;
:onProperty RO:hasPart_County ],
  
   a :Restriction;
:minCardinality "\n\t\t\t\n\t\t"^^xsd:nonNegativeInteger;
:onProperty RO:hasPart_County ],
   
   a :Restriction;
:maxCardinality "\n\t\t\t\n\t\t"^^xsd:nonNegativeInteger;
:onProperty RO:hasPart_County ] .

RO:Product_BusinessObject a :Class;
  rdfs:subClassOf RO:BUSINESS_OBJECT,
    
   a :Restriction;
:allValuesFrom RO:STRING;
:onProperty RO:has_ID ],
    
   a :Restriction;
:cardinality "\n\t\t\n\t\t"^^xsd:nonNegativeInteger;
:onProperty RO:has_ID ],
    
   a :Restriction;
:allValuesFrom RO:FLOAT;
:onProperty RO:has_UnitPrice ],
    
   a :Restriction;
:cardinality "\n\t\t\n\t\t"^^xsd:nonNegativeInteger;
:onProperty RO:has_UnitPrice ],
    
   a :Restriction;
:allValuesFrom RO:FLOAT;
:onProperty RO:has_Height ],
    
   a :Restriction;
:minCardinality "\n\t\t\n\t\t"^^xsd:nonNegativeInteger;
:onProperty RO:has_Height ],
    
   a :Restriction;
:maxCardinality "\n\t\t\n\t\t"^^xsd:nonNegativeInteger;
:onProperty RO:has_Height ],
  
}
a :Restriction;
  :allValuesFrom RO:FLOAT;
  :onProperty RO:has_Width ],
  [ a :Restriction;
    :minCardinality "%n\t\t0\n\t\t"^^xsd:nonNegativeInteger;
    :onProperty RO:has_Width ],
  [ a :Restriction;
    :maxCardinality "%n\t\t1\n\t\t"^^xsd:nonNegativeInteger;
    :onProperty RO:has_Width ],
  [ a :Restriction;
    :allValuesFrom RO:STRING;
    :onProperty RO:has_ProductModel ],
  [ a :Restriction;
    :minCardinality "%n\t\t0\n\t\t"^^xsd:nonNegativeInteger;
    :onProperty RO:has_ProductModel ],
  [ a :Restriction;
    :maxCardinality "%n\t\t1\n\t\t"^^xsd:nonNegativeInteger;
    :onProperty RO:has_ProductModel ],
  [ a :Restriction;
    :allValuesFrom RO:FLOAT;
    :onProperty RO:has_Length ],
  [ a :Restriction;
    :minCardinality "%n\t\t0\n\t\t"^^xsd:nonNegativeInteger;
    :onProperty RO:has_Length ],
  [ a :Restriction;
    :maxCardinality "%n\t\t1\n\t\t"^^xsd:nonNegativeInteger;
    :onProperty RO:has_Length ].

RO:PurchaseOrder_BusinessObjectDocument a :Class;
  rdfs:subClassOf RO:BUSINESS_OBJECT_DOCUMENT,
  [ a :Restriction;
    :cardinality "%n\t\t1\n\t\t"^^xsd:nonNegativeInteger;
    :onProperty RO:has_IssueDate ],
  [
a :Restriction;
:minCardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:hasPart_OrderLine ],
  
  a :Restriction;
:cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:relTo_Supplier ],
  
  a :Restriction;
:cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:relTo_Buyer ],
  
  a :Restriction;
:cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:has_DeliveryDate ],
  
  a :Restriction;
:allValuesFrom RO:STRING;
:onProperty RO:has_ID ],
  
  a :Restriction;
:cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:has_ID ],
  
  a :Restriction;
:allValuesFrom RO:FLOAT;
:onProperty RO:has_TotalPrice ],
  
  a :Restriction;
:cardinality "\n\t\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:has_TotalPrice ],
  
  a :Restriction;
:allValuesFrom [ 
    a :Class;
    :oneOf  ( 
      RO:ExWorks
      RO:FreeCarrier
      RO:FreeOnBoard
      RO:CostAndFreight  ) ];
:onProperty RO:has_TransportationTerms ],
  
  a :Restriction;
:cardinality "\n\t\n\t"^^xsd:nonNegativeInteger;
:onProperty RO:has_TransportationTerms ] .

RO:RELATEDNESS a :ObjectProperty .

RO:STRING a :Class;
  rdfs:subClassOf [ 
    a :Restriction;
    :cardinality "\n\t\n\t"^^xsd:nonNegativeInteger;

RO:Supplier_BusinessActor a :Class;
  rdfs:label "Seller";
  rdfs:subClassOf RO:Organization_BusinessActor .

RO:booleanValue a :DatatypeProperty;
  rdfs:domain RO:BOOLEAN;
  rdfs:range xsd:boolean .

RO:floatValue a :DatatypeProperty;
  rdfs:domain RO:FLOAT;
  rdfs:range xsd:float .

RO:hasPart_BuildingNumber a :ObjectProperty;
  rdfs:domain RO:PostalAddress_ComplexAttribute;
  rdfs:subPropertyOf RO:DECOMPOSITION .

RO:hasPart_City a :ObjectProperty;
  rdfs:domain RO:PostalAddress_ComplexAttribute;
  rdfs:subPropertyOf RO:DECOMPOSITION .

RO:hasPart_CountryCode a :ObjectProperty;
  rdfs:domain RO:PostalAddress_ComplexAttribute;
  rdfs:subPropertyOf RO:DECOMPOSITION .

RO:hasPart_CountryName a :ObjectProperty;
  rdfs:domain RO:PostalAddress_ComplexAttribute;
  rdfs:subPropertyOf RO:DECOMPOSITION .

RO:hasPart_County a :ObjectProperty;
  rdfs:domain RO:PostalAddress_ComplexAttribute;
  rdfs:subPropertyOf RO:DECOMPOSITION .
RO:hasPart_Day a :ObjectProperty;
   rdfs:domain RO:DateTime_ComplexAttribute;
   rdfs:subPropertyOf RO:DECOMPOSITION .

RO:hasPart_Hour a :ObjectProperty;
   rdfs:domain RO:DateTime_ComplexAttribute;
   rdfs:subPropertyOf RO:DECOMPOSITION .

RO:hasPart_Minute a :ObjectProperty;
   rdfs:domain RO:DateTime_ComplexAttribute;
   rdfs:subPropertyOf RO:DECOMPOSITION .

RO:hasPart_Month a :ObjectProperty;
   rdfs:domain RO:DateTime_ComplexAttribute;
   rdfs:subPropertyOf RO:DECOMPOSITION .

RO:hasPart_OrderLine a :ObjectProperty;
   rdfs:domain RO:PurchaseOrder_BusinessObjectDocument;
   rdfs:range RO:OrderLine_BusinessObjectDocument;
   rdfs:subPropertyOf RO:DECOMPOSITION .

RO:hasPart_PostOfficeBoxNumber a :ObjectProperty;
   rdfs:domain RO:PostalAddress_ComplexAttribute;
   rdfs:subPropertyOf RO:DECOMPOSITION .

RO:hasPart_PostalCode a :ObjectProperty;
   rdfs:domain RO:PostalAddress_ComplexAttribute;
   rdfs:subPropertyOf RO:DECOMPOSITION .

RO:hasPart_Second a :ObjectProperty;
   rdfs:domain RO:DateTime_ComplexAttribute;
   rdfs:subPropertyOf RO:DECOMPOSITION .

RO:hasPart_StreetName a :ObjectProperty;
   rdfs:domain RO:PostalAddress_ComplexAttribute;
   rdfs:subPropertyOf RO:DECOMPOSITION .

RO:hasPart_Year a :ObjectProperty;
   rdfs:domain RO:DateTime_ComplexAttribute;
   rdfs:subPropertyOf RO:DECOMPOSITION .

RO:has_DeliveryDate a :ObjectProperty;
   rdfs:domain RO:PurchaseOrder_BusinessObjectDocument;
rdfs:range RO:DeliveryDate_ComplexAttribute;
  rdfs:subPropertyOf RO:ATTRIBUTES .

RO:has_FaxNumber   a :ObjectProperty;
  rdfs:domain RO:BusinessPartner_BusinessActor;
  rdfs:subPropertyOf RO:ATTRIBUTES .

RO:has_Height      a :ObjectProperty;
  rdfs:domain RO:Product_BusinessObject;
  rdfs:subPropertyOf RO:ATTRIBUTES .

RO:has_ID          a :ObjectProperty;
  rdfs:domain [ a :Class;
  :unionOf ( RO:Product_BusinessObject
           RO:PurchaseOrder_BusinessObjectDocument ) ];
  rdfs:subPropertyOf RO:ATTRIBUTES .

RO:has_IssueDate   a :ObjectProperty;
  rdfs:domain RO:PurchaseOrder_BusinessObjectDocument;
  rdfs:range RO:IssueDate_ComplexAttribute;
  rdfs:subPropertyOf RO:ATTRIBUTES .

RO:has_Length      a :ObjectProperty;
  rdfs:domain RO:Product_BusinessObject;
  rdfs:subPropertyOf RO:ATTRIBUTES .

RO:has_LinePrice   a :ObjectProperty;
  rdfs:domain RO:OrderLine_BusinessObjectDocument;
  rdfs:subPropertyOf RO:ATTRIBUTES .

RO:has_PartyTaxID  a :ObjectProperty;
  rdfs:domain RO:BusinessPartner_BusinessActor;
  rdfs:subPropertyOf RO:ATTRIBUTES .

RO:has_PersonFirstName a :ObjectProperty;
  rdfs:domain RO:ContactPerson_BusinessActor;
  rdfs:subPropertyOf RO:ATTRIBUTES .

RO:has_PersonSurname a :ObjectProperty;
  rdfs:domain RO:ContactPerson_BusinessActor;
  rdfs:subPropertyOf RO:ATTRIBUTES .
RO:has_PostalAddress  a :ObjectProperty;
   rdfs:domain RO:BusinessPartner_BusinessActor;
   rdfs:range RO:PostalAddress_ComplexAttribute;
   rdfs:subPropertyOf RO:ATTRIBUTES .

RO:has_ProductModel  a :ObjectProperty;
   rdfs:domain RO:Product_BusinessObject;
   rdfs:subPropertyOf RO:ATTRIBUTES .

RO:has_Quantity  a :ObjectProperty;
   rdfs:domain RO:OrderLine_BusinessObjectDocument;
   rdfs:subPropertyOf RO:ATTRIBUTES .

RO:has_TelephoneNumber  a :ObjectProperty;
   rdfs:domain RO:BusinessPartner_BusinessActor;
   rdfs:subPropertyOf RO:ATTRIBUTES .

RO:has_TotalPrice  a :ObjectProperty;
   rdfs:domain RO:PurchaseOrder_BusinessObjectDocument;
   rdfs:subPropertyOf RO:ATTRIBUTES .

RO:has_TransportationTerms  a :ObjectProperty;
   rdfs:domain RO:PurchaseOrder_BusinessObjectDocument;
   rdfs:subPropertyOf RO:ATTRIBUTES .

RO:has_URI  a :ObjectProperty;
   rdfs:domain RO:BusinessPartner_BusinessActor;
   rdfs:subPropertyOf RO:ATTRIBUTES .

RO:has_UnitPrice  a :ObjectProperty;
   rdfs:domain RO:Product_BusinessObject;
   rdfs:subPropertyOf RO:ATTRIBUTES .

RO:has_Width  a :ObjectProperty;
   rdfs:domain RO:Product_BusinessObject;
   rdfs:subPropertyOf RO:ATTRIBUTES .

RO:has_emailAddress  a :ObjectProperty;
   rdfs:domain RO:BusinessPartner_BusinessActor;
   rdfs:subPropertyOf RO:ATTRIBUTES .

RO:intValue  a :DatatypeProperty;
rdfs:domain RO:INT;
rdfs:range xsd:int .

RO:inverseOf_hasPart_BuildingNumber a :ObjectProperty;
   :inverseOf RO:hasPart_BuildingNumber .

RO:inverseOf_hasPart_City a :ObjectProperty;
   :inverseOf RO:hasPart_City .

RO:inverseOf_hasPart_CountryCode a :ObjectProperty;

RO:inverseOf_hasPart_CountryName a :ObjectProperty;
   :inverseOf RO:hasPart_CountryName .

RO:inverseOf_hasPart_County a :ObjectProperty;
   :inverseOf RO:hasPart_County .

RO:inverseOf_hasPart_Day a :ObjectProperty;
   :inverseOf RO:hasPart_Day .

RO:inverseOf_hasPart_Hour a :ObjectProperty;
   :inverseOf RO:hasPart_Hour .

RO:inverseOf_hasPart_Minute a :ObjectProperty;
   :inverseOf RO:hasPart_Minute .

RO:inverseOf_hasPart_Month a :ObjectProperty;
   :inverseOf RO:hasPart_Month .

RO:inverseOf_hasPart_OrderLine a :ObjectProperty;
   :inverseOf RO:hasPart_OrderLine .

RO:inverseOf_hasPart_PostOfficeBoxNumber a :ObjectProperty;
   :inverseOf RO:hasPart_PostOfficeBoxNumber .

RO:inverseOf_hasPart_PostalCode a :ObjectProperty;
   :inverseOf RO:hasPart_PostalCode .

RO:inverseOf_hasPart_Second a :ObjectProperty;
   :inverseOf RO:hasPart_Second .

RO:inverseOf_hasPart_StreetName a :ObjectProperty;
:inverseOf RO:hasPart_StreetName .

RO:inverseOf_hasPart_Year a :ObjectProperty;
 :inverseOf RO:hasPart_Year .

RO:inverseOf_has_DeliveryDate a :ObjectProperty;
 :inverseOf RO:has_DeliveryDate .

RO:inverseOf_has_FaxNumber a :ObjectProperty;
 :inverseOf RO:has_FaxNumber .

RO:inverseOf_has_Height a :ObjectProperty;
 :inverseOf RO:has_Height .

RO:inverseOf_has_ID a :ObjectProperty;
 :inverseOf RO:has_ID .

RO:inverseOf_has_IssueDate a :ObjectProperty;
 :inverseOf RO:has_IssueDate .

RO:inverseOf_has_Length a :ObjectProperty;
 :inverseOf RO:has_Length .

RO:inverseOf_has_LinePrice a :ObjectProperty;
 :inverseOf RO:has_LinePrice .

RO:inverseOf_has_PartyTaxID a :ObjectProperty;
 :inverseOf RO:has_PartyTaxID .

RO:inverseOf_has_PersonFirstName a :ObjectProperty;
 :inverseOf RO:has_PersonFirstName .

RO:inverseOf_has_PersonSurname a :ObjectProperty;
 :inverseOf RO:has_PersonSurname .

RO:inverseOf_has_PostalAddress a :ObjectProperty;
 :inverseOf RO:has_PostalAddress .

RO:inverseOf_has_ProductModel a :ObjectProperty;
 :inverseOf RO:has_ProductModel .

RO:inverseOf_has_Quantity a :ObjectProperty;
 :inverseOf RO:has_Quantity .
RO:inverseOf_has_TelephoneNumber a :ObjectProperty;
   :inverseOf RO:has_TelephoneNumber.

RO:inverseOf_has_TotalPrice a :ObjectProperty;
   :inverseOf RO:has_TotalPrice.

RO:inverseOf_has_TransportationTerms a :ObjectProperty;
   :inverseOf RO:has_TransportationTerms.

RO:inverseOf_has_URI a :ObjectProperty;
   :inverseOf RO:has_URI.

RO:inverseOf_has_UnitPrice a :ObjectProperty;
   :inverseOf RO:has_UnitPrice.

RO:inverseOf_has_Width a :ObjectProperty;
   :inverseOf RO:has_Width.

RO:inverseOf_has_emailAddress a :ObjectProperty;
   :inverseOf RO:has_emailAddress.

RO:inverseOf_relTo_Buyer a :ObjectProperty;
   :inverseOf RO:relTo_Buyer.

RO:inverseOf_relTo_ContactPerson a :ObjectProperty;
   :inverseOf RO:relTo_ContactPerson.

RO:inverseOf_relTo_Product a :ObjectProperty;
   :inverseOf RO:relTo_Product.

RO:inverseOf_relTo_Supplier a :ObjectProperty;
   :inverseOf RO:relTo_Supplier.

RO:relTo_Buyer a :ObjectProperty;
   rdfs:domain RO:PurchaseOrder_BusinessObjectDocument;
   rdfs:range RO:Buyer_BusinessActor;
   rdfs:subPropertyOf RO:RELATEDNESS.

RO:relTo_ContactPerson a :ObjectProperty;
   rdfs:domain RO:BusinessPartner_BusinessActor;
   rdfs:range RO:ContactPerson_BusinessActor;
   rdfs:subPropertyOf RO:RELATEDNESS.
RO:relTo_Product a :ObjectProperty;
   rdfs:domain RO:OrderLine_BusinessObjectDocument;
   rdfs:range RO:Product_BusinessObject;
   rdfs:subPropertyOf RO:RELATEDNESS .

RO:relTo_Supplier a :ObjectProperty;
   rdfs:domain RO:PurchaseOrder_BusinessObjectDocument;
   rdfs:range RO:Supplier_BusinessActor;
   rdfs:subPropertyOf RO:RELATEDNESS .

RO:stringValue a :DatatypeProperty;
   rdfs:domain RO:STRING;
   rdfs:range xsd:string .

<http:/iasi.cnr.it/#RO.owl> a :Ontology .

#ENDS
B2. FSA Example

In this section is presented the FSA associated to ds-elem
buyerInfo.buyerInfo_has_organisationInfo.organisationInfo.has_contactPerson.contactPerson.has_name:string
extracted from AIDIMA document previously presented.

D2.1. Translation of PSA into OWL

This subsection presents the N3 code (derived from Owl code) associated to each path defined in PSAV.

\[
\text{PSAV} = \\
[ \text{Buyer\_BusinessActor\_relTo\_ContactPerson\_ContactPerson\_BusinessActor\_has\_PersonFirstName:STRING,} \\
\text{Buyer\_BusinessActor\_relTo\_ContactPerson\_ContactPerson\_BusinessActor\_has\_PersonSurname:STRING} ] \\
\]

- n3 code to express the first path in PSAV:

\[
\text{ROA:FSAPath1 a :Class;}
\text{rdfs:subClassOf [}
\text{ a :Class;}
\text{intersectionOf (}
\text{ROA:STRING [}
\text{ a :Class;}
\text{rdfs:equivalentClass [}
\text{ a :Restriction;}
\text{onProperty <#inverseOf\_ has\_PersonFirstName>;}
\text{someValuesFrom [}
\text{ a :Class;}
\text{intersectionOf (}
\text{ROA:ContactPerson\_BusinessActor [}
\text{ a :Class;}
\text{rdfs:equivalentClass [}
\text{ a :Restriction;}
\text{onProperty <#inverseOf\_ relTo\_ContactPerson>;}
\]

- n3 code to express the second path in PSAV:

\[
\text{ROA:FSAPath2 a :Class;}
\text{rdfs:subClassOf [}
\text{ a :Class;}
\text{intersectionOf (}
\text{ROA:STRING [}
\text{ ] ] ) ]
\]
D2.2. Translation of SSA into OWL

In this section is proposed a way to express an SSA expression into OWL. This approach proposes a way to translate the existence of the abstract operators (⊕, ϕ) using the owl:annotationProperty construct.

In particular, the example concerns the annotation of ds-elem:

buyerInfo.buyerInfo_has_organisationInfo.organisationInfo.has_contactPerson.contactPerson.has_name:string

The following SSA:

SSA(buyerInfo.buyerInfo_has_organisationInfo.organisationInfo.has_contactPerson.contactPerson.has_name:string) ·
Buyer_BusinessActor.relTo.ContactPerson.ContactPerson_BusinessActor.has_PersonFirstName:STRING ⊕
Buyer_BusinessActor.relTo.ContactPerson.ContactPerson_BusinessActor.has_PersonSurname:STRING

can be translated into following owl code:

<owl:AnnotationProperty rdf:ID="plus_operator"/>
<owl:AnnotationProperty rdf:ID="composes"/>

<owl:Class rdf:ID="FSAPath1">
  <rdfs:subClassOf> <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#STRING"/>
      <owl:Class rdf:equivalentClass> <owl:Restriction>
        <owl:onProperty rdf:resource="#inverseOf_has_PersonFirstName"/>
        <owl:someValuesFrom> <owl:Class>
          <owl:intersectionOf rdf:parseType="Collection">
            a :Class;
rdfs:equivalentClass [a :Restriction;
            :onProperty <#inverseOf_ has_PersonSurname>;
            :someValuesFrom [a :Class;
            :intersectionOf {
              ROA:ContactPerson_BusinessActor
              [a :Class;
                rdfs:equivalentClass [a :Restriction;
                  :onProperty <#inverseOf_relTo_ContactPerson>;
Analogously is for the second path (FSAPath2).
Programme
Integrating and Strengthening the European Research
Strategic Objective
Networked businesses and governments

Integrated Project / Programme Title
Advanced Technologies for Interoperability of Heterogeneous Enterprise Networks and their Application

Acronym
ATHENA

Project No
507849

ATHENA – Project Name
Knowledge Support and Semantic Mediation Solutions

ATHENA A - Project No
A3

<table>
<thead>
<tr>
<th>Deliverable Nr: D.A3.3</th>
</tr>
</thead>
</table>

Semantic Annotation language and tool for Information and Business Processes

Appendix E: Annotation Repository

Work Package – A3

Leading Partner: CNR-IASI

Security Classification: Project Participants (PP)

February 14, 2006

Version 1.0
Table of contents

APPENDIX E: ANNOTATION REPOSITORY

E1. ANNOTATION REPOSITORY DATA STRUCTURES 1
E2. ANNOTATION REPOSITORY WEB SERVICE SPECIFICATION 2

Table of Figures

Figure 1 – Annotation Repository data structures................................................................................... 1
Appendix E: Annotation Repository

E1. Annotation Repository data structures

In this section the diagrams of classes included in annotation repository are shown:

![Diagram of Annotation Repository data structures]

Figure 1 – Annotation Repository data structures
E2. Annotation Repository web service specification

Below, the WSDL document of the Annotation Repository web service is reported.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="AnnotRepos" xmlns="http://schemas.xmlsoap.org/wsdl/
xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="AnnotRepos" xmlns:intf="AnnotRepos"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/

<wsdl:types>
<schema targetNamespace="http://dataStructure.webServices.astar"
xmlns="http://www.w3.org/2001/XMLSchema">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/
<complexType name="Job">
<sequence>
<element name="masterAuthor" nillable="true" type="xsd:string"/>
<element name="ontologyId" nillable="true" type="xsd:string"/>
<element name="resourceId" nillable="true" type="xsd:string"/>
</sequence>
</complexType>
<complexType name="Annotation">
<sequence>
<element name="annotationLevel" nillable="true" type="xsd:string"/>
<element name="annotationExpr" nillable="true" type="xsd:string"/>
<element name="connective" nillable="true" type="xsd:string"/>
<element name="annotationId" nillable="true" type="xsd:string"/>
</sequence>
</complexType>
<complexType name="AnnotatedResPath">
<sequence>
<element name="resPathId" nillable="true" type="xsd:string"/>
<element name="annotations" nillable="true" type="impl:ArrayOf_tns1_Annotation"/>
</sequence>
</complexType>
</schema>
<wsdl:types>
<schema targetNamespace="AnnotRepos" xmlns="http://www.w3.org/2001/XMLSchema">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/
<complexType name="ArrayOf_tns1_Annotation">
<complexContent>
<restriction base="soapenc:Array">
<attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:Annotation[]"/>
</restriction>
</complexContent>
</complexType>
<complexType name="ArrayOf_tns1_AnnotatedResPath">
<complexContent>
<restriction base="soapenc:Array">
<attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:AnnotatedResPath[]"/>
</restriction>
</complexContent>
</complexType>
</schema>
<wsdl:types>
<wsdl:message name="getAnnotationsByJobIdRequest">
<wsdl:part name="in0" type="xsd:string"/>
</wsdl:message>
</wsdl:definitions>
```
<wsdl:message name="getJobByIdResponse">
  <wsdl:part name="getJobByIdReturn" type="tns1:Job"/>
</wsdl:message>

<wsdl:message name="getAnnotationByLevelResponse">
  <wsdl:part name="getAnnotationByLevelReturn" type="tns1:Annotation"/>
</wsdl:message>

<wsdl:message name="getAnnotationsByLevelResponse">
  <wsdl:part name="getAnnotationsByLevelReturn" type="impl:ArrayOf_tns1_AnnotatedResPath"/>
</wsdl:message>

<wsdl:message name="getAnnotationsByResPathIdResponse">
  <wsdl:part name="getAnnotationsByResPathIdReturn" type="impl:ArrayOf_tns1_Annotation"/>
</wsdl:message>

<wsdl:message name="getAnnotationsByJobIdResponse">
  <wsdl:part name="getAnnotationsByJobIdReturn" type="impl:ArrayOf_tns1_AnnotatedResPath"/>
</wsdl:message>

<wsdl:message name="getAnnotationsByResPathIdRequest">
  <wsdl:part name="in0" type="xsd:string"/>
  <wsdl:part name="in1" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="getAnnotationsByLevelRequest">
  <wsdl:part name="in0" type="xsd:string"/>
  <wsdl:part name="in1" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="getAnnotationByLevelRequest">
  <wsdl:part name="in0" type="xsd:string"/>
  <wsdl:part name="in1" type="xsd:string"/>
  <wsdl:part name="in2" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="getJobByIdRequest">
<wsdl:part name="in0" type="xsd:string"/>

</wsdl:message>

<wsdl:portType name="AnnotationRepository">

<wsdl:operation name="getJobById" parameterOrder="in0">
    <wsdl:input message="impl:getJobByIdRequest" name="getJobByIdRequest"/>
    <wsdl:output message="impl:getJobByIdResponse" name="getJobByIdResponse"/>
</wsdl:operation>

<wsdl:operation name="getAnnotationsByJobId" parameterOrder="in0">
    <wsdl:input message="impl:getAnnotationsByJobIdRequest" name="getAnnotationsByJobIdRequest"/>
    <wsdl:output message="impl:getAnnotationsByJobIdResponse" name="getAnnotationsByJobIdResponse"/>
</wsdl:operation>

<wsdl:operation name="getAnnotationsByLevel" parameterOrder="in0 in1">
    <wsdl:input message="impl:getAnnotationsByLevelRequest" name="getAnnotationsByLevelRequest"/>
    <wsdl:output message="impl:getAnnotationsByLevelResponse" name="getAnnotationsByLevelResponse"/>
</wsdl:operation>

<wsdl:operation name="getAnnotationsByResPathId" parameterOrder="in0 in1">
    <wsdl:input message="impl:getAnnotationsByResPathIdRequest" name="getAnnotationsByResPathIdRequest"/>
    <wsdl:output message="impl:getAnnotationsByResPathIdResponse" name="getAnnotationsByResPathIdResponse"/>
</wsdl:operation>

<wsdl:operation name="getAnnotationByLevel" parameterOrder="in0 in1 in2">
    <wsdl:input message="impl:getAnnotationByLevelRequest" name="getAnnotationByLevelRequest"/>
    <wsdl:output message="impl:getAnnotationByLevelResponse" name="getAnnotationByLevelResponse"/>
</wsdl:operation>

</wsdl:portType>

<wsdl:binding name="AnnotationRepositorySoapBinding" type="impl:AnnotationRepository">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getJobById">
        <wsdlsoap:operation soapAction=""/>
    </wsdl:operation>
</wsdl:binding>
<wsdl:input name="getJobByIdRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="AnnotRepos" use="encoded"/>
</wsdl:input>

<wsdl:output name="getJobByIdResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="AnnotRepos" use="encoded"/>
</wsdl:output>

<wsdl:operation name="getAnnotationsByJobId">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="getAnnotationsByJobIdRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="AnnotRepos" use="encoded"/>
  </wsdl:input>
  <wsdl:output name="getAnnotationsByJobIdResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="AnnotRepos" use="encoded"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="getAnnotationsByLevel">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="getAnnotationsByLevelRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="AnnotRepos" use="encoded"/>
  </wsdl:input>
  <wsdl:output name="getAnnotationsByLevelResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="AnnotRepos" use="encoded"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:operation name="getAnnotationsByResPathId">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="getAnnotationsByResPathIdRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="AnnotRepos" use="encoded"/>
  </wsdl:input>
  <wsdl:output name="getAnnotationsByResPathIdResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="AnnotRepos" use="encoded"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:input name="getAnnotationsByResPathIdRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="AnnotRepos" use="encoded"/>
</wsdl:input>

<wsdl:output name="getAnnotationsByResPathIdResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="AnnotRepos" use="encoded"/>
</wsdl:output>
</wsdl:operation>

<wsdl:operation name="getAnnotationByLevel">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="getAnnotationByLevelRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="AnnotRepos" use="encoded"/>
  </wsdl:input>
</wsdl:operation>

<wsdl:service name="AnnotationRepositoryService">
  <wsdl:port binding="impl:AnnotationRepositorySoapBinding" name="AnnotationRepository">
    <wsdlsoap:address location="http://localhost:8080/axis/services/AnnotationRepository"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
Programme
Integrating and Strengthening the European Research
Strategic Objective
Networked businesses and governments

Integrated Project / Programme Title
Advanced Technologies for Interoperability of Heterogeneous Enterprise Networks and their Application

Acronym
ATHENA

Project No
507849

ATHENA – Project Name
Knowledge Support and Semantic Mediation Solutions
ATHENA - Project No
A3

Deliverable Nr: D.A3.3

Semantic Annotation language and tool for Information and Business Processes

Appendix F: User Manual

Work Package – A3

Leading Partner: CNR-IASI

Security Classification: Project Participants (PP)

February 14, 2006

Version 1.0
Table of contents

F.1 EXECUTIVE SUMMARY
   Introduction 1
   Restriction of use 1
   Access to the system 1
   Disclaimer 1

F.2 INTRODUCTION TO SEMANTIC ANNOTATION
   The A* Annotation Methodology 2
   Semantic Mismatches 2
   Four steps to a Full Semantic Annotation. 3
   Summary of the Annotation methodology 5

F.3 THE A* SYSTEM
   Generality 7
   A* User categories 7
   Job organization 7
   A* Login 7
   Creating a new Job 8
   Loading an existing Job 9
   The A* main window 11

THE SEMANTIC ANNOTATION DEFINITION
   The Terminological Annotation 14
   The Path Annotation 17
   Simple Annotation 20
   Full Annotation 21
   How to end an annotation session 21

Table of Figures

Figure 1 – Multilevel Semantic Annotation ................................................................. 6
Figure 2 - Login window ................................................................................................ 8
Figure 3 - New Job creation window .......................................................................... 9
Figure 4 - Save Job window ........................................................................................ 9
Figure 5 - Load Job window ....................................................................................... 10
Figure 6 – An example of loaded job ......................................................................... 11
Figure 7 – Ontology Panel display modes (Classes, Properties and Property Paths) ... 12
Figure 8 – Loaded annotated paths ........................................................................... 13
Figure 9 – Activation of the Terminological Annotation level .................................. 14
Figure 10 – Terminological Annotation: proposal of annotation ............................... 15
Figure 11 – Terminological Annotation: removing and adding terms ...................... 16
Figure 12 – Path Annotation: Creating an annotation .............................................. 18
Figure 13 – Path Annotation 2: Saving an annotation ............................................... 19
Figure 14 – An example of Simple Annotation ......................................................... 20
Figure 15 - An example of Full Annotation ............................................................... 21
F.1 Executive summary

This document represents the A* V1.0 User Manual The purpose of this document is to provide a guidance to the use of the A* system.

Introduction

A* System is part of the Athena Semantic suite, as described in the DA3.3 documentation; A* is the A3 tool to allow the Semantic Annotation of electronic documents and software resources to full-fill the objectives of Electronic Interoperability, as stated in the Athena project.

The current document is composed of the following chapters:

• Chapter 1: is the present executive summary;
• Chapter 2: provides a summary of the semantic annotation methodology, as implemented by the A* System;
• Chapter 3: is devoted to the description of the fundamental interactions that the A* System allows the user to perform;
• Chapter 4: provides additional details on the different levels of Semantic Annotation, that the user can perform with A*

The user, wishing to obtain a deeper understanding of the A* functions and architectural components, should refer to the Athena Deliverable DA3.3 basic documentation.

Restriction of use

A* has been developed and released as a deliverable of the European ATHENA project A3, Knowledge Support and Semantic Mediation Solutions. The A* tool is released for use by the authorized Athena project members for purposes linked to the Athena project objectives; any other use outside this project should be authorized by the Athena Project management.

Access to the system

The A* system functionalities can be accessed, via a common web browser, at the address http://leks-pub.iasi.cnr.it/Astar. Potential users must request in advance access rights to the A* Administration Center by e-mail to: osimi@iasi.cnr.it.

Disclaimer

A* V1.0 is a proof of concept product and is released as a prototype software within an advanced research program; LEKS, IASI-CNR cannot be held responsible for loss of data or information through potential malfunctions of the released software, even if great care has been taken in A* development and testing.

LEKS, IASI-CNR welcomes any information and suggestions that may improve the system. Improvements and communications can be addressed to: osimi@iasi.cnr.it.
F.2 Introduction to Semantic Annotation

The current document is the user guide of the A*, a software tool developed by LEKS at IASI-CNR, for the definition of ontology-based semantic annotations. The current chapter provides a brief introduction to the Semantic Annotation methodology, as it has been implemented in the A* tool.

The A* Annotation Methodology

This section illustrates the main features of the A* method for semantic annotation (SA) and the checking of the alignment of a Resource Schema, with respect to a given Reference Ontology. Such a method is based on a stepwise semantic annotation approach, structured on four different steps, where the elements of a Resource Schema are associated with ontology-based expressions of increasing complexity.

The semantic annotation intends to solve some definition mismatches between the Resource Schema and the reference ontology. In this way, through the use of appropriate semantic annotations, the meaning of the elements composing the Resource Schema can be made non ambiguous.

Semantic Mismatches

In DA3.3 we identified a limited number of possible mismatches. Such mismatches categories are sketchily recapped in the table below [Table 1]. The examples reported in the table are drawn from an eProcurement business scenario, proposed in the Athena project, and aim at representing the different cases in an informal, intuitive way. In particular, we consider a PurchaseOrder business document and a corresponding reference ontology. In our example we assume, that the Document Schema is represented in the RDFS1/N3 notation and the Reference Ontology in OWL/N3 notation.

As shown in the table, the possible mismatches have been divided into two broad categories: lossless and lossy mismatches. Lossless mismatches are cases in which annotation can fully capture the intended semantics of the annotated element, while lossy mismatches represent cases where a semantic preserving mapping to the reference ontology cannot be built.

<table>
<thead>
<tr>
<th>Lossless mismatches</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>Naming</td>
<td>different labels for the same concept</td>
</tr>
<tr>
<td>Attribute Granularity</td>
<td>the same information is decomposed into a different number of attributes (or sub-attributes)</td>
</tr>
<tr>
<td>Structuring</td>
<td>different design structures in organising the information</td>
</tr>
<tr>
<td>SubClass-Attribute</td>
<td>an attribute, with a predefined value set, is represented by a set of subclasses, one for each value</td>
</tr>
</tbody>
</table>

1 without loss of generality, we assume a lossless transformation of a document from, say, XSD to RDFS
2 A formal treatment of the semantics falls outside the scope of the paper. Here the semantics is addressed at an intuitive level.
Four steps to a Full Semantic Annotation.

The proposed method for semantic annotation is based on a stepwise approach that allows an annotation to be progressively refined. It starts from the simplest form, a vector of keywords, and progressively evolves, through 4 different stages, till it reaches the last level, represented by an OWL expression. Here we briefly introduce the four steps of our approach, and we indicate the sorts of mismatches that can be identified at each step. In the next section, the four steps will be extensively illustrated.

Step 1 – Terminological Semantic Annotation (TSA)

In this step the terms used in the DS are contrasted with the terminology of the RO (i.e. the
“rdf:id” and the “rdf:label” used for the ontology elements) to find terminological matches. The annotation is then achieved in the form of vectors of RO terms. In particular, the reported ontology terms correspond to the concepts preferred terms, that are reported under the “id” element in the ontology (unless otherwise indicated by the user).

During this step, the following mismatches are singled out:

- **Naming mismatch**
- **Coverage mismatch**
- **Abstraction mismatch**

**Step 2 – Path Semantic Annotation (PSA)**

In this step, the terms identified in the TSA are used to achieve a first annotation that considers the structures of both: document and ontology. Therefore, terms are composed to form paths. Here we consider in particular property paths, i.e., that paths that start from an entity and connect it to one of its atomic properties (i.e., typed with a basic type: integer, string, etc.) For instance,

```
Order.has_orderHeader.orderHeader.has_buyerInfo.buyerInfo.buyerInfo_has_organisationInfo.organisationInfo.has_phone:string
```

is a property path.

We focus on property paths since atomic properties are the key elements used in data exchange (i.e., they actually carry data). But another important issue is that a single term is often not sufficient to unambiguously express the intended semantics. For instance, the property `has_phone` may represent information associated to `Buyer_BusinessActor` or to `SellerBusinessActor` depending on the paths in which it participates.

Each document path is associated to one or more RO paths, that represent the PSA. The matching of the single elements of the paths are performed by considering the mappings achieved in the Step 1.

In this step, the following kind of semantic mismatches are identified:

- **Structural mismatch.**
- **SubClass-Attribute mismatch**
- **Schema-Instance mismatch**
- **Content mismatch.**

In the first two steps the method produces semantic annotations represented by vectors of RO terms and RO paths (semantic annotation vector, sav) that are associated to individual terms and paths of the business document, respectively. As illustrated above, the use of semantic annotation vectors is useful to identify a significant number of semantic mismatches. However, we need to progress further in
the refinement of the annotations, introducing semantic annotation expressions (sax) that make use of specific operators.

**Step 3 – Simple Semantic Annotation (SSA)**

In this step, the ontology paths, identified in the previous step, are used to compose Simple expressions by using abstract operators. An SSA expression is typically a composition of RO paths that better captures the intended meaning of the annotated DS element. This level has the specific aim to facilitate the reconciliation phase.

In this step, the following kinds of semantic mismatches are identified:

- **Attribute Granularity mismatch.**
- **Encoding mismatch**
- **Precision mismatch.**

**Step 4 – Full Semantic Annotation (FSA)**

In this step, a simple annotation is refined and coded into OWL. Every path that appears in the simple annotation expression is translated into OWL DL code in automatic way. In this way, we have specified the annotation expression, in a formally grounded ontology representation language.

It is important to repeat that each step refines the precision of the annotation of previous level. Every Semantic Annotation assumes the following form:

\[ ds-elem =: sav | sax \]

in case the semantic annotation vector (sav) or the semantic annotation expression (sax) fully captures the intended meaning of the annotated element. Otherwise, we have the following situations (and the corresponding connectives):

- \( ds-elem >: sav | sax \) (Overspecification), to express that the annotated ds-elem has a richer intended semantics than the associated annotation expression or vector;
- \( ds-elem <: sav | sax \) (Underspecification), to express that the annotated ds-elem has an intended semantics poorer than the associated annotation expression or vector;
- \( ds-elem partOf sav | sax \), to express that a de-elem represents a component of the structured concept identified by the annotation expression or vector.

**Summary of the Annotation methodology**

In Figure 1 a summary of the four levels of annotation is sketchily reported. For each annotation level, a brief description and the corresponding user skill level, able to build the annotation, are
indicated. We distinguish four kinds of user skill level: (i) business person, with a basic domain knowledge but without technical (knowledge representation) competencies, (ii) domain expert, with a sound domain knowledge but without technical (knowledge representation) competencies, (iii) domain expert, with a blend of domain knowledge and technical competencies, (iv) knowledge engineering, with full competences in ontology modelling, but with a limited domain knowledge.

![Multilevel Semantic Annotation](image)

**Figure 1 – Multilevel Semantic Annotation**
F.3 The A* system

The A* software system has been developed, as part of DA3.3, by LEKS, IASI-CNR within the Athena Project, to support the definition and management of ontology-base semantic annotation.

Generality

The system is accessible via a web browser (in the current V1.0 version only through Internet Explorer). All the system functions can be performed remotely.

The A* system server, with the reference ontology repository, is centrally managed by an Administrator, whose responsibilities are to perform all the activities related to the A* system maintenance and use.

A* User categories

The A* system recognizes the following user categories:

- The A* Administrator, is in charge of users management, for instance, granting new users accounts;
- The A* User, has access to all the functionalities of the system in order to define semantic annotation of documents/models.

Only the A* User functionalities are described in the current document.

Job organization

With the A* tool Semantic annotations are applied to a document schema or model (resource schema in the following paragraphs) by reference to a specific domain ontology.

A new Job is created for the annotation of a specific resource schema; a Job is identified by the definition of a resource schema and a domain ontology and represent an annotation activity on a specific resource schema through the use of a specific ontology.

A* Login

In order to be able to access the A* system each potential user must previously obtain from the A* Administrator a user ID and an associated password.

The A* system is accessible via the Internet at the address http://leks-pub.iasi.cnr.it/Astar. After connection with the A* home-page, the A* Login window appears (Figure 2), allowing the user to access to A* by filling:

- the User Name field: with the user ID assigned by the Administrator;
- the Password field: the related password assigned by the Administrator.
Figure 2 - Login window

(Format and Language into the window is browser dependent).

The OK button allows the user to submit the inserted data and, if correctly identified, to enter. If user name or password are wrong, the following Unidentified user message appears:

“You are not authorized to access this resource”

Creating a new Job

After the log in, if the user intends to annotate a new resource schema, he has to create a new Job. Selecting the New Job item in the File menu, the New Job creation window will appear (Figure 3). The window will show the list of the accessible ontologies (the ones previously created with the Athos Ontology Management System) and the list of the accessible resource schemata (with the current A* V1.0 the resource schemata should have been previously stored in the Themis repository). In order to create a new Job, the user has to select an ontology and a resource schema and click on the Create Job button to confirm the action. After that, in order to store the Job just created on the database, the Job has to be saved. In order to do that, the user must select the Save Job item in the File menu, and the Save Job window will appear (Figure 4). In this window, the user must specify a name for the Job and click the Save button..
Loading an existing Job

In order to modify or edit an existing Job the user has to select the **Load Job** in the **File Menu**, and the **Load an existing Job** window will appear (Figure 5). The window shows the list of the existing Jobs, accessible by the current user; by selecting one entry from the list, the corresponding Job will be loaded.
When the Job has been loaded, the main window of A* will be opened and filled with the data corresponding to the loaded Job (Figure 6).
The A* main window

The A* main window is composed by:
- an upper banner which includes the Athena Logo, the A* logo and the LEKS, IASI-CNR logo;
- the menu bar through which it is possible to access the main functionalities of the system;
- The **Ontology Panel**, on the left, in which the ontology content is shown. The ontology panel has three display modes (Figure 7):
  - **Classes**: in this mode the main concepts whose kind is Actor, Object, Process and Complex Attribute are shown. They are identified by a **orange** icon;
  - **Properties**: in this mode the properties involving the concepts are listed in the following way:
    - Properties linking concepts to Atomic Attributes are identified by an **blue** icon;
    - Properties linking concepts to non Atomic Attributes are identified by an **green** icon.
  - **Property Paths**: in this mode, for each concept in the ontology, the paths starting from this concept and ending in an Atomic Attribute are listed.

The display mode can be selected by pointing the mouse on the corresponding label and clicking the left button. On the display of the ontology panel each single element (Classes, Properties and Paths) can be selected by using the right button of the mouse.
Figure 7 – Ontology Panel display modes (Classes, Properties and Property Paths)

- The **Resource Panel**, where the resource schema is shown, is situated in the right upper part of the screen. The resource schema is represented in a UML Class diagram-like view. In this panel there are three small buttons:
  - **Move Boxes**: when this button is selected, boxes in the panel can be moved.
  - **Pick Path**: when this button is selected, a path can be selected
  - **Clear Path**: when this button is selected the currently selected path is de-activated (deselected).

- The **Annotation Panel**, where the annotation are defined, is situated in the right lower part of the screen. Since A* supports 4 levels of annotation, depending on the chosen level, the Annotation Panel will assume a different appearance.

After having loaded an existing Job, the paths from the Resource Schema, that have been previously annotated, are highlighted in blue in the Resource Panel (Figure 8).

The current path from the Resource schema (the one on which the user is currently working) is highlighted in red.
Figure 8 – Loaded annotated paths
The Semantic Annotation definition

According to the 4 levels of annotation methodology, on which the A* tool is based, 4 main functionalities and environments are supplied; they are described in the following paragraphs.

The Terminological Annotation

In order to activate the Terminological Annotation, the user has to switch to this level, by selecting the Terminological entry in the Annotation->Level menu (Figure 9). The Annotation Panel switches to the Terminological mode. In this mode, the Annotation Panel is divided into two vertical sub-panels:

- A left sub-panel, which shows all the terms involved in the resource schema;
- A right sub-panel which, for each term from the resource schema, shows the terms from the ontology associated to it.

Since the terminological annotation level is supported by a semi-automatic functionality, when an user access, for the first time, the terminological annotation of a Job, the system, will suggest a proposal of annotation (Figure 10).
Modifying a proposal of Terminological Annotation

The terminological annotation proposed by the system, can be modified by the user, by adding and/or removing terms from the right sub-panels (Figure 11).

- To remove a specific term the user has to click on the green button on the right of the term, switching the button to red;
- To add a term (that was not considered by the automatic matching functionality), the user has to select the new term by clicking with the right button on the Ontology Panel; the new term will be added at the bottom of the list of the proposed terms.
Figure 11 – Terminological Annotation: removing and adding terms
The Path Annotation

In order to activate the Path Annotation, the user has to select the Path entry in the Annotation-Level menu. The Annotation Panel switches to the Path mode. In this mode, the Annotation Panel is divided into two horizontal sub-panels:

- the upper sub-panel for selecting a path from the resource schema;
- the lower panel for selecting a vector of paths from the ontology.

Selecting a path from the resource schema

To select a path from the resource schema, the user has to interact with the Resource Panel.

- Select the Pick Path button, switching it to red;
- Select the box that will represent the root of the path; the colour of the box will switch to red;
- Select the relations that identify the path; the boxes involved are automatically selected. The colour of both, boxes and relations, switches to red.

After the path has been highlighted in the Resource Panel, by clicking the Get Path button in the Annotation Panel a new line item in the upper part of the Annotation Panel will show the selected path. By clicking on it, the lower part of the Annotation Panel will be cleared and one or more paths can be selected from the ontology.

Selecting paths from the ontology

To select one or more paths from the ontology, the user has to interact with the Ontology Panel in the following way:

- Switch the Ontology panel to the Property Paths view;
- Identify the path that has to be selected;
- Select the ending element of the path. In this way, the full path, from the root to the ending element, will switch the colour to red;
- Click the Get Path button in the lower sub-panel of the Annotation panel (Figure 12); the selected ontology path will be added to the vector of paths created in the lower sub-panel of the annotation panel.

Selecting a connective

Furthermore, to define an annotation at Path level, it is necessary to specify a connective between the path from the Resource schema and the vector of paths from the ontology.

In the lower sub-panel of the Annotation panel, a pull down menu, allows the user to chose the proper connective. The connective options are the following:

- “=” which specifies that the selected paths from the ontology, exactly match the path from the Resource Schema;
- “>” which specifies that the selected paths from the ontology, under-specify the path from the Resource Schema;
- “<” which specifies that the selected paths from the ontology, over-specify the path from the Resource Schema.
Figure 12 – Path Annotation: Creating an annotation

After having defined the Annotation, the user can saved it by clicking on the Save button in the Annotation panel. A message will show that the annotation has been stored (Figure 13).
Figure 13 – Path Annotation 2: Saving an annotation

All the annotated paths from the Resource Schema will be listed in the upper sub-panel of the Annotation panel.

By clicking on one of it with the left button of the mouse; the corresponding annotation is shown in the sub-panel below.
Simple Annotation

In order to activate the Simple Annotation, the user has to select the Simple item in the Annotation->Level menu (Figure 14). The Annotation Panel switches to the Simple mode. In this mode, the Annotation Panel, like in the Path mode, is divided into two horizontal sub-panels:

- the upper sub-panel for selecting a path from the resource schema;
- the lower panel for selecting a vector of paths from the ontology:

![Simple Annotation](image)

The construction of a Simple Annotation is similar to the construction of the Path Annotation.

With respect to the Path Annotation, a Simple Annotation identifies an expression built on paths from the ontology. For that purpose, two abstract operators are available:

- $\phi$: an unary operator. It works on a single path from the ontology. It can be used for example for solving Encoding mismatches.
- $+$: a binary operator. It can be used for instance, for solving Attribute Granularity mismatches

An example of Simple Annotation is shown in Figure 14.
Full Annotation
At this level the annotation is represented by an OWL expression.

In order to activate the Full Annotation, the user has to select the Full item in the Annotation- >Level menu. The Annotation Panel switches to the Full mode. Also in this mode, the Annotation Panel is divided into two horizontal sub-panels:

- the upper sub-panel for selecting a path from the resource schema;
- the lower panel for displaying the DL expression corresponding to the Full annotation.

As previously said in section 0, due to limitations of the OWL-DL expressive power, only Path annotations are translated into OWL. Furthermore, since the OWL/XML syntax is too verbose the Full annotations are represented in DL expressions.

An example of Full Annotation is shown in Figure 15.

How to end an annotation session

After an annotation session, the user can save the job by selecting the Save Job item in the File menu.
Programme
Integrating and Strengthening the European Research
Strategic Objective
Networked businesses and governments
Integrated Project / Programme Title
Advanced Technologies for Interoperability of Heterogeneous Enterprise Networks and their Application
Acronym
ATHENA
Project No
507849
ATHENA – Project Name
Knowledge Support and Semantic Mediation Solutions
ATHENA A - Project No
A3

**Deliverable Nr: D.A3.3**

Semantic Annotation language and tool for Information and Business Processes

Appendix G: Installation Guide

**Work Package – A3**

Leading Partner: CNR-IASI

Security Classification: Project Participants (PP)

February 14, 2006

Version 1.0
Table of contents

G.1 REQUIREMENTS TO INSTALL A* TOOL 1
G.2 A* TOOL INSTALLATION 1
G.3 A* INSTALLATION TEST 1
G.1 Requirements to install A* tool

- Plone (you can use http://plone.org/products/plone/releases/2.1.1)
- adobe svg viewer (you can use http://services.txt.it/argos/installation/SVGView.exe)

G.2 A* Tool installation

Just suppose that:
- Plone has been installed in the following path: C:\Programs\Plone 2
- Athos has been installed

To install the A* tool on your machine you have to perform the following steps:
1. copy the "yapps2" and "rdflib" folders in the directory C:\Programs\Plone 2\Python\Lib\site-package
2. copy the "ZImportAnnot" folder in the directory C:\Programs\Plone 2\Zope\lib\python\Products
3. copy "Astar.zexp" file in the folder C:\Programs\Plone 2\Zope\import
4. run Plone by launching Plone from start -->programs-->Plone 2,
   o access with the internet browser the "http://localhost:<portNumber>/manage_main" page and click the "Import/Export" button
   o write "Astar.zexp" in the textArea named "Import file name" and select the "Take ownership of imported objects" option

G.3 A* installation test

. Open your browser (Internet Explorer) and insert the following address:
   http://localhost:8080/Astar

If the installation has been completed successfully the A* HOME PAGE will appear.
**Programme**
Integrating and Strengthening the European Research
**Strategic Objective**
Networked businesses and governments

**Integrated Project / Programme Title**
Advanced Technologies for Interoperability of Heterogeneous Enterprise Networks and their Application

**Acronym**
ATHENA

**Project No**
507849

**ATHENA – Project Name**
Knowledge Support and Semantic Mediation Solutions

**ATHENA A - Project No**
A3

---

**Deliverable Nr: D.A3.3**

Semantic Annotation language and tool for Information and Business Processes

**AppendixH** An approach for integration of A3 solutions and A6 models repository

**Work Package – A3**

Leading Partner: CNR-IASI

Security Classification: Project Participants (PP)

February 14, 2006

Version 1.0
Table of contents

H1 INTEGRATING THE A3 SOLUTIONS IN THE A6 SEMANTICALLY ENHANCED MODELS
REPOSITORY ARCHITECTURE
  Architectural integration of the A6 model repository and the A3 semantic annotation solution 1
  A brief consideration on the A* architecture 3
  The XMI Class diagram transformation into the A* internal representation 4

Table of Figures

Figure 1 - Integration between the A6 enhanced model repository architecture and the A3 solutions . 2
Figure 2: Resource document and ontology acquisition and visualization ............................................. 4
Figure 3: Example of UML Class diagram .............................................................................................. 5
H1 Integrating the A3 solutions in the A6 semantically enhanced models repository architecture

This annex wants to report on how to integrate the A3 results in terms of semantic annotation, with the A6 models repository, in order to give an additional approach to the semantic enhancement of such a repository.

DA3.3 has been dedicated to the description of an ontology-based semantic annotation method and tool, A*, and it has been described how such method and tool can be applied to different kinds of digital resources.

In the DA3.3 it has been described that, independently of the kind of resource that is being treated by the A* tool, such a resource (or schema document) is transformed in an internal representation. In this way, both the visualization of the schema document to be annotated and the whole annotation process, are made independent of the format of the specific kind of resource that is being considered.

The DA3.3 has been mainly described taking into consideration RDFS as the format of the schema document. Of course, such an approach derives from the fact that the A* annotation tool has been mainly described in the context of the semantic reconciliation, where the considered documents are represented by using RDF(S).

For this purpose, in the Annex A of the DA3.3, it has been described how RDFS documents are transformed into the A* internal representation. Such internal representation is then used for two main objectives:

- Visualize the schema document in the A* graphical user interface,
- Represent the semantic annotations.

As previously said, such an internal representation of the schema document is very important because it allows to make the semantic annotation process, independent of the format of the specific document that is being annotated.

Now, considering the fact that the A6 model repository allows to export such models into the XMI format, the most part of this annex will be dedicated to describe how to map the XMI Class diagram representation of a model, into the A* internal representation of a schema document.

Nevertheless, the first part of the rest of this annex will be dedicated to the architectural issues regarding the integration of the A6 architecture for semantic enhancement of model repository and the A3 semantic annotation solution.

Architectural integration of the A6 model repository and the A3 semantic annotation solution

Figure 6 in the DA6.2 describes the architecture of a semantically enhanced model repository. Figure 1 here, clearly refers to such a picture, with the intent of positioning the result of the A3 semantic annotation solution into the A6 semantically enhanced model repository architecture.

At a first glance, it is clear that on the architectural point of view the integration fits well. This is due to the fact that both the generic architecture of the A6 approaches,
and the A3 approach, consider the complete separation between the model to be annotated and the relative annotation. Both in fact separates the model repository from the annotation repository.

In particular, respect to the Figure 6 of the DA6,2 in Figure 1:

- the generic annotation tool has been replaced by the A* semantic annotation tool,
- the Annotation Service has been replaced by the A* annotation service, the ASSERT query engine and a link to the Athos ontology management system. More in detail:
  - the knowledge base has been replaced by the A* Annotation Repository service and the relative Annotation Repository;
  - the ontology involvement, by the link with the Athos Ontology Repository service,
  - the Rules engine and reasoner by the ASSERT semantic query engine.

Figure 1 - Integration between the A6 enhanced model repository architecture and the A3 solutions
A brief consideration on the A* architecture

As previously said at the beginning of this annex, one of the important issue for an integration of the A3 semantic solutions and the A6 semantically enhanced model repository architecture, is the capability of the A* tool, the A3 semantic annotation tool, to manage (to annotate) the type of models managed by the A6 models repository.

The A6 models repository is a MOF repository and it provides an XMI interface. In particular it means that such MOF models can be exported in the XMI format.

On this basis, the idea is that the A* tool acquires the XMI representation of the stored MOF model and allows to define semantic annotations on it. Of course, the correspondence between the XMI export and the MOF internal representation inside the models repository is guaranteed by the XMI import/export functions.

In order to be as independent as possible of the type (format) of resource to be annotated, the architecture of the A* tool has been conceived for being very modular in this sense. As reported in the DA3.3, the main steps for allowing the acquiring the schema document and allowing the annotation are achieved by two modules:

- the **docLoader**, that is in charge of acquiring the document
- the **ResViz**, which is in charge of providing the graphical visualization of the document.

The job of the docLoader depends on the specific models (schema documents) repository and the type of format of such models, while the job of the ResViz module does not. In fact, once has acquired the document schema, it transforms the document into an internal representation, that is the same for all the types of format. Such an internal representation is passed to the ResViz service that can produce the visualization of the document to be annotated. Figure 2 reports the picture of part of the A* architecture that involves the mentioned modules.

What has been said right now, clarifies the reason and the importance of the internal representation in A* of the document to be annotates.

For this reason, in the following, the transformation of an XMI document into the A* internal representation will be described.
Figure 2: Resource document and ontology acquisition and visualization

The XMI Class diagram transformation into the A* internal representation

In this section, the transformation of an XMI Class diagram document into the A* internal representation of a schema document will be outlined.

The first step of this activity is an intermediate transformation that transforms the original XMI document into an XMI-Light document. XMI-Light\(^1\) is an XML format conceived by SINTEF for representing an XMI document in a more concise way. In particular, the XMI-Light format does not consider the information regarding the positioning of the graphical element.

Let’s consider the UML Class diagram in Figure 3, the XMI document representing such a simple diagram is thousands of lines of code long; the code is not reported for reasons of space. Nevertheless, the corresponding transformation of the XMI document into XMI-Light is reported below.; this is very concise and easier to read.

Such a transformation can be obtained automatically by using the UMT tool developed by SINTEF.

\(^1\) J. Oldevik, *UMT documentation*. 22-03-2004
TransportationMean

SerialNumber: String

Car

SerialNumber: String
drivenBy

Person

Wheel

Engine

Figure 3: Example of UML Class diagram

```xml
  <package id="I4e75bcm109444676bdmm7f55" name="model 1" xmlns:UML1_1="UML1.1" xmlns:UML1_2="UML1.2">
    <class abstract="false" id="I4e75bcm109444676bdmm7f51" name="Car" superClass="I4e75bcm109444676bdmm7ef2">
      <attribute cardinality="" id="I4e75bcm109444676bdmm7f3a" name="Color" type="I4e75bcm109444676bdmm7f29"/>
      <association aggregationType="none" cardinality="1..1" id="I4e75bcm109444676bdmm7f15" name="drivenBy" otherCardinality="1..1" targetClass="I4e75bcm109444676bdmm7f28"/>
      <association aggregationType="composite" cardinality="1..1" id="I4e75bcm109444676bdmm7e7b" name="MissingRoleName" otherCardinality="1..1" otherEnd="I4e75bcm109444676bdmm7e7e"/>
    </class>
  </package>
</model>
```
<association aggregationType="composite" cardinality="1..1" id="I4e75bcm109444676bdmm7e58" name="MissingRoleName" otherCardinality="1..1" otherEnd="I4e75bcm109444676bdmm7e5b" targetClass="I4e75bcm109444676bdmm7ec7" />
</class>

<class abstract="false" id="I4e75bcm109444676bdmm7f28" name="Person" />
<class abstract="false" id="I4e75bcm109444676bdmm7ef2" name="TransportationMean" >
<attribute cardinality="" id="I4e75bcm109444676bdmm7ed8" name="SerialNumber" type="I4e75bcm109444676bdmm7f29" />
</class>
<class abstract="false" id="I4e75bcm109444676bdmm7ec7" name="Engine" >
<association aggregationType="none" cardinality="1..1" id="I4e75bcm109444676bdmm7e5b" name="MissingRoleName" otherCardinality="1..1" otherEnd="I4e75bcm109444676bdmm7e58" targetClass="I4e75bcm109444676bdmm7f51" />
</class>
<class abstract="false" id="I4e75bcm109444676bdmm7eb4" name="Wheel" >
<association aggregationType="none" cardinality="1..1" id="I4e75bcm109444676bdmm7e7e" name="MissingRoleName" otherCardinality="1..1" otherEnd="I4e75bcm109444676bdmm7e7b" targetClass="I4e75bcm109444676bdmm7f51" />
Since the document to be annotated will be shown in the A* graphical user interface into an UML Class Diagram like view, the internal representation needs to describe at least Classes, Associations and Attributes. That is done as described in the following:

- **Classes** are represented through a list of pairs of the following format:
  
  \[(\text{class\_Id}, \text{class\_Label})\]
  
  where \text{class\_Id} and \text{class\_Label} correspond respectively to the identifier and the name of the Class.

  Respect to the XMI-Light document above, we will have the following pairs representing the Classes:

  - ("I4e75bcm109444676bdmm7ef2", "TransportationMean")
  - ("I4e75bcm109444676bdmm7f51", "Car")
  - ("I4e75bcm109444676bdmm7eb4", "Wheel")
  - ("I4e75bcm109444676bdmm7ec7", "Engine")
  - ("I4e75bcm109444676bdmm7f28", "Person")
  - ("I4e75bcm109444676bdmm7f29", "String")

- **Specialization** are represented through a list of triples of the following format:

  \[((\text{classS\_Id}, \text{classS\_Label}), (\text{classD\_Id}, \text{classD\_Label}))\]
where classS and classD represent respectively the source (the more specific Class) and the destination (the more general Class) classes of the specialization and assoc represents the association linking the two classes.

Respect to the XMI-Light document above, we will have just one tuple representing the Specialization between Car and TransportationMean.

(("I4e75bcm109444676bdmm7ef2", "TransportationMean"),
("I4e75bcm109444676bdmm7f51", "Car"))

- **Associations** are represented through a list of triples of the following format:

  $$((classS\_Id, classS\_Label), (assoc\_Id, assoc\_Label, "Association"), (classD\_Id, classD\_Label))$$

  where classS and classD represent respectively the source and the destination classes of the association and assoc represents the association linking the two classes.

  Respect to the XMI-Light document above, we will have just one tuple representing the drivenBy Association.

  (("I4e75bcm109444676bdmm7f51", "Car"),
  ("I4e75bcm109444676bdmm7f15", "drivenBy", "Association"),
  (I4e75bcm109444676bdmm7f28, "Person"))

- **Aggregation** are represented through a list of triples of the following format:

  $$((classS\_Id, classS\_Label), (aggreg\_Id, "Aggregation"), (classD\_Id, classD\_Label))$$

  where classS and classD represent respectively the source and the destination classes of the aggregation and aggreg\_Id represents the association linking the two classes.

  Respect to the XMI-Light document above, we will have two tuples for the Aggregation involving the Car Class with the Wheel and the Engine Classes

  (("I4e75bcm109444676bdmm7f51", "Car"),
  ("I4e75bcm109444676bdmm7e58", "Aggregation"),
  ("I4e75bcm109444676bdmm7e7b", "Engine"))

  (("I4e75bcm109444676bdmm7f51", "Car"),
  ("I4e75bcm109444676bdmm7eb4", "Wheel"))

- **Attributes** are represented through a list of triples of the following format:

  $$((class\_Id, class\_Label), (attr\_Id, attr\_Label, "Attribute"), (basicType\_Id, basicType\_Label))$$
where the Class `class` has an Attribute `attr` whose type is `basicType` (i.e., int, string, ...).

Respect to the XMI-Light document above, we will have just one tuple representing the `SerialNumber` Attribute.

```plaintext
(("I4e75bcm109444676bdmm7ef2", "TransportationMean"),
 ("I4e75bcm109444676bdmm7ed8", "SerialNumber", "Attribute"),
 ("I4e75bcm109444676bdmm7f29", "String"))
```