A Knowledge Integration Framework for Robotics.

Jacob Persson, Axel Gallois, Anders Björkelund, Love Hafdell, Mathias Haage, Jacek Malec, Klas Nilsson, Pierre Nugues
Department of Computer Science, Lund University, Sweden

Abstract

This paper describes a knowledge integration framework for robotics, whose goal is to represent, store, adapt, and distribute knowledge across engineering platforms. The architecture abstracts the components as data sources, where data are available in the AutomationML data exchange format. AutomationML is an on-going standard initiative that aims at unifying data representation and APIs used by engineering tools. A triplification procedure converts native formats used by data sources into RDF triples and then exposes them via a SPARQL endpoint. The triplification step has been implemented for the CAEX top level and logic data parts of AutomationML, where the conversion uses XSLT rules.

1 Introduction

Although widely used in industrial production, robots are still restricted to large series and require highly-skilled workforce to carry out the preproduction steps: design, setup, and programming. A crucial element in the design is the composition of the production platform and its configuration. This composition is often made complex because of the heterogeneity of the equipments and engineering tools that use specific data formats and need adaptation layers. Adaptation stacks are often completely specific to one product and one production setup.

This paper attempts to solve problems arising from disparate data formats or representations between equipments using a semantic *knowledge integration framework* (KIF). We describe an architecture to represent, store, adapt, and distribute knowledge used in robotized production, where we abstract the robotics components as data sources. In our robotics applications, we assume that data is available in the AutomationML exchange format [9]. AutomationML is an on-going standard initiative that aims at unifying data representation and APIs used by engineering tools. Using the scheme proposed by DBpedia [1, 2], we implemented a procedure that converts native formats used by data sources into RDF triples and then exposes them via a SPARQL endpoint.

This architecture makes it easier to integrate knowledge and skills used in robotized production and access them using a uniform protocol. It should simplify and streamline both the design and the composition of a robotics cell, as well as the implementation of adaptation layers between equipments.

2 Categories of Knowledge Used in Robotized Production

Knowledge in robotics systems covers a considerable set of disciplines and categories: logic information, finite–state machines and discrete-event systems, differential-algebraic systems, geometric and kinematics models, databases and first-order logic, and robot task programs to name a few.

All these categories use different data representations and for a same category, the formats of engineering tools may also be different. This lack of standardization is of course a problem to manufacturers, as they cannot easily switch equipments, and causes well-known difficulties to customize the production.

2.1 AutomationML

AutomationML[1] is a standardized markup language that attempts to model and unify all kinds of information used by engineering tools. It covers plant topology, geometry and kinematics, logic information, reference and relations, and referencing other formats [9]. Rather than reinventing representations, we started from AutomationML as it integrates existing data formats and provides the glue to tie them together.

The upper-level part of AutomationML uses the CAEX data exchange format. CAEX is a framework to store hierarchical object information, properties, and libraries [10]. It represents topology information in the form of plants, cells, components, attributes, interfaces, relations, and references [3]. This CAEX top-level connects the different data formats used downstream by the different categories of engineering tools; for example COLLADA [4] for geometry and kinematics data and PLCopen [15] for logic data.

2.2 Logic Data in AutomationML

AutomationML aims at comprehensively describing most of the fields of automation and robotics. Logic data [13] is one of them and AutomationML provides a representation for four types of logic models: Gantt charts, PERT charts, impulse diagrams, and sequential function charts. AutomationML uses PLCopen as target format and introduces a bridge format, the intermediate modeling layer

[1] http://www.automationml.org/

(IML), to transform specific models or proprietary encodings into PLCopen. AutomationML carries out the conversion of nonnative formats in two steps: The first step takes the original format and produces an IML representation; the second one produces a PLCopen XML document from the intermediate representation.

2.3 Extracting Information from AutomationML and PLCopen

The XML format used in AutomationML and PLCopen is not well suited for semantic processing or reasoning as its structure is far from any kind of query or rule formalism. From this viewpoint, the code is difficult to grasp and this makes it complex to extract meaningful information easily. Gantt charts are one example given in the AutomationML/PLCopen specifications [13, pp.72–84]. Data we may want to extract from them are the predecessor to a given task and the condition of their completion, for instance. Even if PLCopen is not targeted to human readers, the XML document structure makes this extraction unintuitive. In addition, it could suffer from some attribute and variable naming discrepancies across different files.

3 Exposing the Semantics of AutomationML

3.1 The Resource Description Framework

The resource description framework (RDF) [16] is a initiative of the World Wide Web consortium (W3C) to bring semantics to the web. The framework represents information as collections of triples consisting of a subject, a predicate, also called a property, and an object.
A collection of triples forms a directed graph, where the predicates correspond to the arc labels and the subjects and objects, the pairs of connected vertices (Fig. 1).

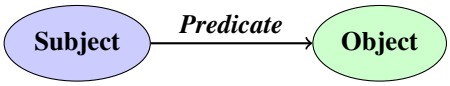

Figure 1: An RDF triple. The predicate is sometimes called the property

Subjects and predicates are unambiguously named using uniform resource identifiers (URIs); objects are either resources or literal values, i.e. numbers, strings, dates, etc. Literal values can only correspond to terminal nodes.
The next line

```
<demozelle#d1e7> caex-xml:hasName "Linie".
```

is an example of a triple that connects the subject node `<demozelle#d1e7>` to a literal string, `"Linie"`. The predicate gives a meaning to this relation: the object is a name. The prefix `caex-xml` is the abbreviation of a complete URI and in our current implementation, it stands for

```
http://asimov.ludat.lth.se/2009/09/caex-xml.owl#
```

The RDF model is close to concepts used in classical logic [8]. In the traditional predicate logic notation, each RDF triple

```
Subject Predicate Object
```

would correspond to the statement (or the fact):

```
Predicate(Subject, Object).
```

In addition to standardization, the possibility to reformulate the RDF model into a logic setting enables its users to benefit from a considerable amount of results and tools.

3.2 RDF Extensions

The W3C defined additional languages or vocabularies on top of RDF. The two most significant ones are RDF schema, RDFS, and the Web ontology language (OWL):

- RDFS defines notably the word `rdfs:Class` that allows to declare a RDF resource as a class and the predicate `rdfs:subClassOf` that allows to declare subclasses of a class and build a hierarchy.
- OWL extends RDFS with a set of relations among concepts, such as equality, inheritance, disjointedness, and transitivity. This allows us to perform simple consistency checks on a collection of triples expressed in OWL, given an ontology.

3.3 Exposing AutomationML as RDF Triples

To expose the semantics of CAEX and PLCopen files and make information extraction easier, we converted them to RDF triples. Following the DBpedia method [1], we implemented a procedure that transforms the data sources used in an AutomationML environment into RDF repositories and we made them accessible using the SPARQL query protocol. The procedure comprises the following steps:

1. Extract and transform data from all the knowledge sources into RDF triples;
2. Expose the resulting graphs and make them accessible using RDF repositories. For some nodes, we used the Linked Data method to associate the node URIs to HTTP accessible data;
3. Access and modify the graphs from a central integration server using a SPARQL update endpoint or another update mechanism.

Figure 2 shows a sketch of the system overall architecture and Fig. 3, an outline of the communication between a data source and the knowledge integration server.

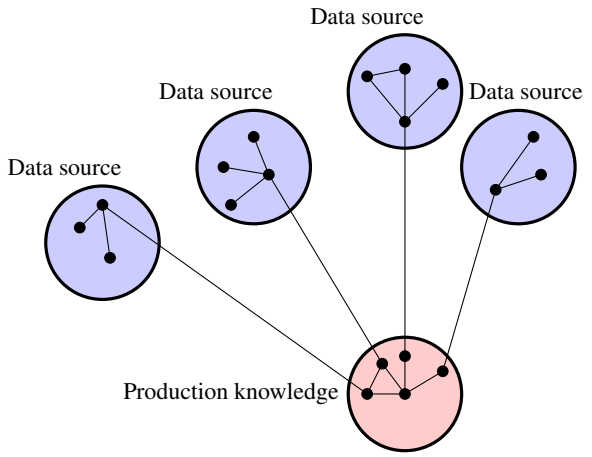

Figure 2: Sketch of the system architecture.

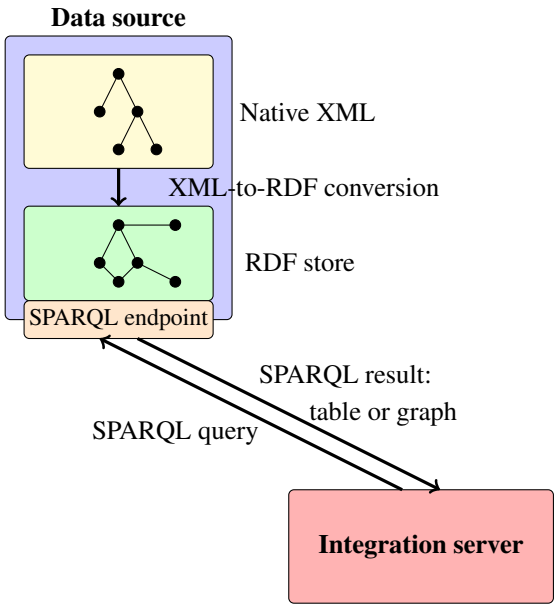

Figure 3: The communication mechanism.

3.4 Converting XML to RDF

Numerous methods to convert XML to RDF have been investigated [12, 14, *inter alia*]. The conversion is not necessarily trivial and may depend on what kind of RDF vocabulary, possibly OWL ontology, is needed by the application. A transformation that would enrich the XML original structure with an ontology and make use of the full OWL capacity is a complex problem. Coming up with a meaningful ontology to describe a complete plant hierarchy and/or logic data and geometry, for example, would need further investigation. For an investigation on rules to use to convert CAEX documents into OWL, see [18].

In our case, we used the *document object model* (DOM) of the original XML documents and we applied a syntactic mapping to produce the RDF triples. The conversion procedure uses rules that take the elements of the source DOM tree (i.e. `xsd:elements` and `xsd:attributes`) and create the corresponding nodes of the output RDF graph. The RDF predicates are automatically generated from the XML element names by the concatenation of the prefix `has` and the element name.

The procedure is implemented in the XSLT language, where XSLT builds a DOM representation of the XML input document, traverses the tree, and applies the rules to produce the resulting RDF graph. The transformations are then straightforward and the graph can easily be understood by the means of the XML schema describing the original document.

Overall, this procedure transforms XML documents – including the AutomationML standard suite – into queryable databases or data sources. A drawback of it is that we lose the ability to validate the consistency of the RDF graph with respect to the underlying XML schema. This could be solved by constructing an OWL ontology and using a reasoner.

3.5 A Conversion Example

The AutomationML specifications require to start the hierarchy with a CAEX top-level. From this level, we can branch other formats such as logic data. As an example, the XML code below shows an excerpt of a description used as the root of a document:

```
<CAEXFile FileName="sockectconnector.aml">
 <InstanceHierarchy Name="SocketConnector">
   <InternalElement Name="Station" ID="{1}">
     <InternalElement Name="Robot" ID="{2}"/>
   </InternalElement>
 </InstanceHierarchy>
 <RoleClassLib
     Name="AutomationMLRoleClassLib">
   <RoleClass Name="AutomationMLBaseRole"/>
 </RoleClassLib>
</CAEXFile>
```

This code can be represented by the DOM tree shown in Fig. 4. Using XSLT conversion rules, we mapped the DOM tree onto an RDF graph as shown in Fig. 5.

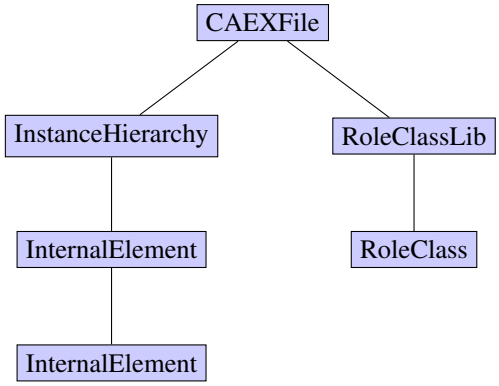

Figure 4: DOM structure of a CAEX document. Attributes are omitted for simplicity.

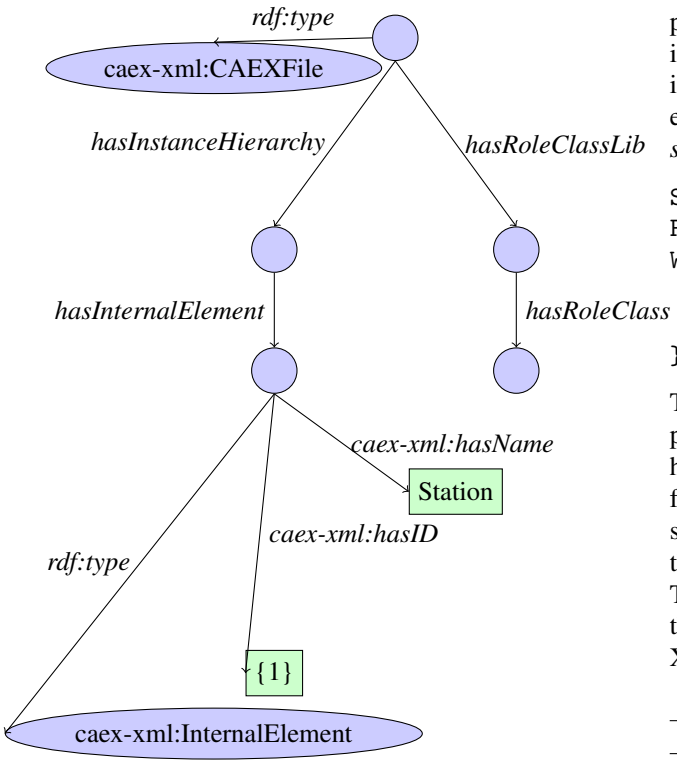

Figure 5: DOM to RDF conversion. For sake of simplicity, only a subset of the nodes and arcs is included in the figure.

4 Hosting RDF Triples

We hosted the converted triples in network-enabled RDF repositories using Sesame servers developed by the OpenRDF[2] community. Sesame is a tool for storing and querying RDF data. RDF repositories are deployed as Java servlet containers with a web interface so that users can easily save and export graphs. In addition to the subject–predicate–object triple, Sesame associates a context to each statement. Such contexts correspond to graph names and take the form of URIs. A repository can thus contain one or more named graphs.
Figure 6 shows the complete architecture of the knowledge integration framework (KIF) and the standard components we used to implement the first prototype.

[2] `http://www.openrdf.org/`

4.1 Querying Triple Stores

Once the data sources are available as RDF repositories, we can use SPARQL [20] to express queries across the graphs. SPARQL extracts patterns from them in a way that is related to Prolog or Datalog [19]. SPARQL can deliver the output as a table using the `SELECT` keyword or format it as an RDF graph using `CONSTRUCT`.
RDF access is possible with Sesame via a SPARQL endpoint, the platform API, or directly using Sesame's web interface. Using the example in Sect. 3.5 and the SPARQL interface, the following query extracts the names of the elements in an instance hierarchy, in our case a *robot* and a *station*:

```
SELECT ?ie ?name
FROM <http://asimov.ludat.lth.se/isr>
WHERE {
  ?ih caex-xml:hasInternalElement ?ie.
  ?ie caex-xml:hasName ?name
}
```

The graph uses the `caex-xml:hasInternalElement` predicate to connect the internal elements to the instance hierarchy. The first line extracts the internal elements `?ie` from an instance hierarchy `?ih` using this predicate. The second line extracts the `caex-xml:hasName` attribute of the internal elements.
The query produces the results shown in Table 1, where the node URIs have been automatically generated by the XML-to-RDF conversion procedure.

?ie	?name
<http://asimov.ludat.lth.se/isr#id0x1ef3eda0>	"Station"
<http://asimov.ludat.lth.se/isr#id0x1ef3bfd0>	"Robot"

Table 1: Results from the SPARQL query.

4.2 Network Architecture

4.2.1 Linked Data

The architecture of the knowledge integration framework is designed to host multiple clients. They include browsers to review engineering data, editors, and simulation tools that may have to interact with the RDF repositories. The architecture also embeds the concept of Linked Data [5],

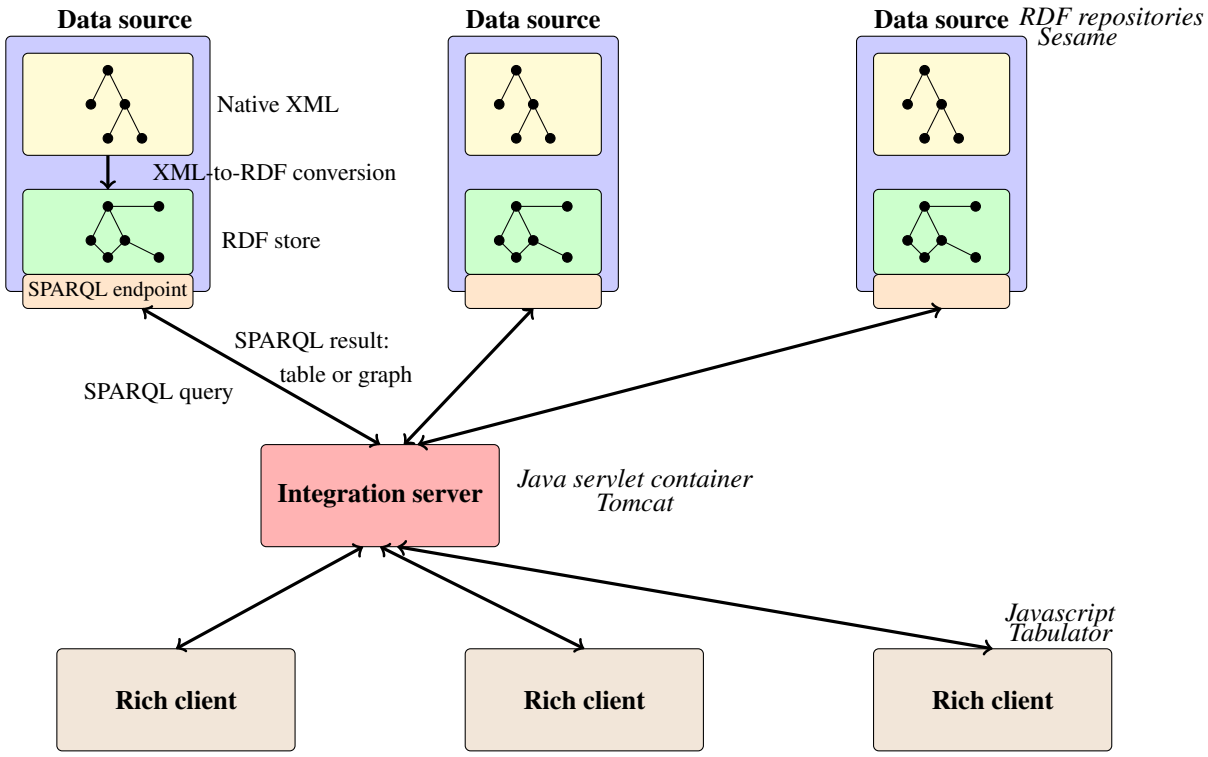

Figure 6: Complete architecture of the knowledge integration framework (KIF) including the visualization clients.

where we associate certain nodes of the graph to a URI with a content accessible via the HTTP protocol.
In the context of AutomationML, CAEX, and their translation into RDFs, this means that we can access class descriptions as they correspond to nodes in the graph. Figure 7 shows an example with the role class *Port* from the AutomationML documentation. All the converted classes could have a similar URL that would link them directly to metadata on their purpose, functionality, etc.

4.2.2 Network Protocol

The W3C has defined specifications of a REST-inspired interface to RDF repositories called the SPARQL protocol [7]. The REST model [11, 17] – representation state transfer – is an architecture to handle communications between clients and servers. REST uses standards from the Web: URI/URL to identify resources being accessed or transferred and HTTP as communication protocol, which makes it easy to understand and implement. For a discussion, see [22].
Using the SPARQL protocol, the query:

```
SELECT ?s ?p ?o
WHERE {?s ?p ?o}
```

is sent to a repository in a HTTP envelope using the `GET` method. This mechanism makes the integration of a client–server architecture easier as tools exist and are well specified.

In practice, HTTP messages can be built using tools like cURL [21]. Using this tool, the command to send the query above is:

```
curl -X GET -H \
"Accept: application/sparql-results+json" \
http://asimov.ludat.lth.se/openrdf-sesame/\
repositories/sandbox?query=SELECT+\
%3fs+%3fp+%3fo+WHERE+%7b%3fs+%3fp+%3fo%7d
```

where `Accept: application/sparql-results+json` tells the server to send the results in JSON, `http://asimov.ludat.lth.se/` is the server, `openrdf-sesame/repositories/sandbox` is the repository, `query` tells that it is the start of the query, and `%3f` is the UTF-8 value – simply ASCII here – of the question mark for instance.
The REST concept provides a uniform network interface to the servers that can return any type of RDF results:

1. Graph results corresponding to SPARQL `CONSTRUCT` queries, retrieval of a specific context, or retrieval of the complete graph in a repository;
2. Tuple results corresponding to SPARQL `SELECT` queries as shown in the example above; and
3. Boolean results corresponding to SPARQL `ASK` queries.

In our current implementation, we used Sesame's implementation of the SPARQL REST protocol to handle data

"Port" a caex-xml:RoleClass

Description	"The role class “Port” is a role type for objects that groups a number of interfaces and allows describing complex interfaces in this way. AutomationML Port objects shall reference this role. Details and examples are specified in 9.2. Additionally, if required, the AutomationML Port object shall have a CAEX ExternalInterface derived from the AutomationML InterfaceClass “PortConnector” (see Table 25). Note: This interface allows connecting the considered Port with a number of other ports on an abstract level without detailed description of the inner relations between the sub-interfaces (see Figure 40)."
Parent	http://asimov.ludat.lth.se/aml/roles/AutomationMLBaseRoleClassLib/AutomationMLBaseRole
Base class	http://asimov.ludat.lth.se/aml/roles/AutomationMLBaseRoleClassLib/AutomationMLBaseRole

"Direction" (this#Direction)

Description	"This attribute shall be used to describe the direction of the Port. Values shall be one of the following: “In” or “Out” or “InOut”. Ports with the direction “In” can only be connected to ports with the direction “Out” or “InOut” and ports with the direction “Out” can only be connected with ports with the direction “In” or “InOut”. Ports with the direction “InOut” can be connected to Ports of arbitrary direction. Examples: Direction = ”Out” (e.g. a plug) Direction = “In” (e.g. a socket) Direction = “InOut” This information can be used e.g. in order to prove the validity of a connection."
Datatype	"xs:string"

Attributes

"Cardinality" (this#Cardinality)

Description	"This attribute is a complex attribute type and shall not have a value. The corresponding sub-attributes are described in Table 24."
Datatype	"xs:complexType"

Nested attributes:

"MinOccur" (this#MinOccur)

Description	"The MinOccur value describes the minimum possible number of connections to or from this Port. The attribute shall have values greater or equal to 0. Example: MinOccur = 1 This means that this Port should be connected with minimum one other Port."
Datatype	"xs:uint"

"MaxOccur" (this#MaxOccur)

Description	"The MaxOccur describes the maximum possible number of connections to or from this Port. The attribute shall have values greater than or equal to MinOccur, except 0 which means infinite. Example: MaxOccur = 3 This means that this Port can only be connected with maximal three other ports."
Datatype	"xs:uint"

"Category" (this#Category)

Description	"The category attribute describes the Port type. The value of this attribute is user-defined. Only ports with the same category value are allowed to be connected. Example: Category = “MaterialFlow”"
Datatype	"xs:string"

Underlying RDF triples

this	rdf:type	caex-xml:RoleClass
this	caex-xml:hasName	"Port"
this	caex-xml:refBaseClassPath	http://asimov.ludat.lth.se/aml/roles/AutomationMLBaseRoleClassLib/AutomationMLBaseRole
this	caex-xml:hasAttribute	this#Direction
this	caex-xml:hasAttribute	this#Cardinality
this	caex-xml:hasAttribute	this#Category
this	caex-xml:hasDescription	"The role class “Port” is a role type for objects that groups a number of interfaces and allows describing complex interfaces in this way. AutomationML Port objects shall reference this role. Details and examples are specified in 9.2. Additionally, if required, the AutomationML Port object shall have a CAEX ExternalInterface derived from the AutomationML InterfaceClass “PortConnector” (see Table 25). Note: This interface allows connecting the considered Port with a number of other ports on an abstract level without detailed description of the inner relations between the sub-interfaces (see Figure 40)."
http://asimov.ludat.lth.se/aml/roles/AutomationMLBaseRoleClassLib/AutomationMLBaseRole	caex-xml:hasRoleClass	this

Figure 7: Linked data to associate nodes of the graph to accessible URIs.

communication between servers and clients within the KIF architecture and manage the RDF graphs.

5 The Visualization Module

The visualization module is the end component of the knowledge integration framework. Requirements for it include portability, concurrent access, and distribution across the internet. We used standard web browsers as the main vehicle for visualization. They fit the description above and do not need specific installations or configurations. AJAX technologies combining asynchronous JavaScript and XML are then a natural choice to develop the client side of the knowledge integration framework; carry out data transfer and processing, page rendering, animation, and interaction.
We developed prototype components to validate the proposed architecture.

- We hosted the triples on Sesame servers.
- A knowledge integration server in the form of a Java servlet container collects RDF data from the distributed stores and carries out additional processing.
- Finally, the rich clients use AJAX techniques to render the results and support user interaction.

Servers and users can be distributed on any point of the internet.
Using the knowledge integration framework, we converted the DemoZelle example that comes with the AutomationML editor available from `http://www.automationml.org/`. We transformed the CAEX description into RDF and designed SPARQL queries to extract information from it. Information is then visualized on the browser using Javascript in a way that is similar to that of the AutomationML editor (Fig. 8).
Objects with a COLLADA model are visualized in the center pane using Java and Java 3D. The lower pane of the KIF viewer enables the user to navigate in the graph. Each click on a node will update the view with nodes connected to the selected node. We built the visualizer with the help of Javascript scripts from the Tabulator project [6].

6 Discussion

As we wrote earlier, the XML formats used in AutomationML and PLCopen may not be well suited for semantic processing or reasoning as their structure may be far from a rule formalism. Since the RDF graph is a transposition of the XML tree, we still face similar problems when dealing with this graph and SPARQL queries could prove more difficult to write, sometimes.
To make more sense out of these graphs, one or more additional ontologies could be created that would still fit the initial RDF graph. This is the approach used by [18] on a limited set of examples. As multiple inheritance is allowed in RDF, this does not contradict the ontology derived from the XML schema, but could be considered as an additional layer on top of the initial XML-like graph.

Acknowledgments

The research leading to these results has received funding from the European Union's seventh framework program (FP7/2007-2013) under grant agreement n° 230902.

References

[1] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In Aberer, editor, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007*, Busan, Korea, November 11-15 2007.

[2] Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, and David Aumueller. Triplify - lightweight linked data publication from relational databases. In *WWW 2009*, 2009.

[3] AutomationML. AutomationML specification. Part 1 – Architecture and general requirements. Technical report, AutomationML Group, January 2009.

[4] Mark Barnes and Ellen Levy Finch. COLLADA – digital asset schema release 1.5.0. Technical report, Khronos Group, 2008.

[5] Tim Berners-Lee. Linked data. `http://www.w3.org/DesignIssues/LinkedData.html`, 2006.

[6] Tim Berners-Lee, Yuhsin Chen, Lydia Chilton, Dan Connolly, Ruth Dhanaraj, James Hollenbach, Adam Lerer, and David Sheets. Tabulator: Exploring and analyzing linked data on the semantic web. In *Proceedings of the The 3rd International Semantic Web User Interaction Workshop*, November 2006.

[7] Kendall Grant Clark, Lee Feigenbaum, and Elias Torres. Sparql protocol for rdf. `http://www.w3.org/TR/rdf-sparql-protocol/`, 2008.

[8] Jos De Bruijn, Enrico Franconi, and Sergio Tessaris. Logical reconstruction of normative RDF. In *OWL: Experiences and Directions Workshop*, 2005.

[9] Rainer Drath, editor. *Datenaustausch in der Anlagenplanung mit AutomationML. Integration von CAEX, PLCopen, XML und COLLADA*. Springer, 2010.

[10] Murat Fedai, Ulrich Epple, Rainer Drath, and Alexander Fay. A metamodel for generic data exchange between various CAE systems. In I. Troch

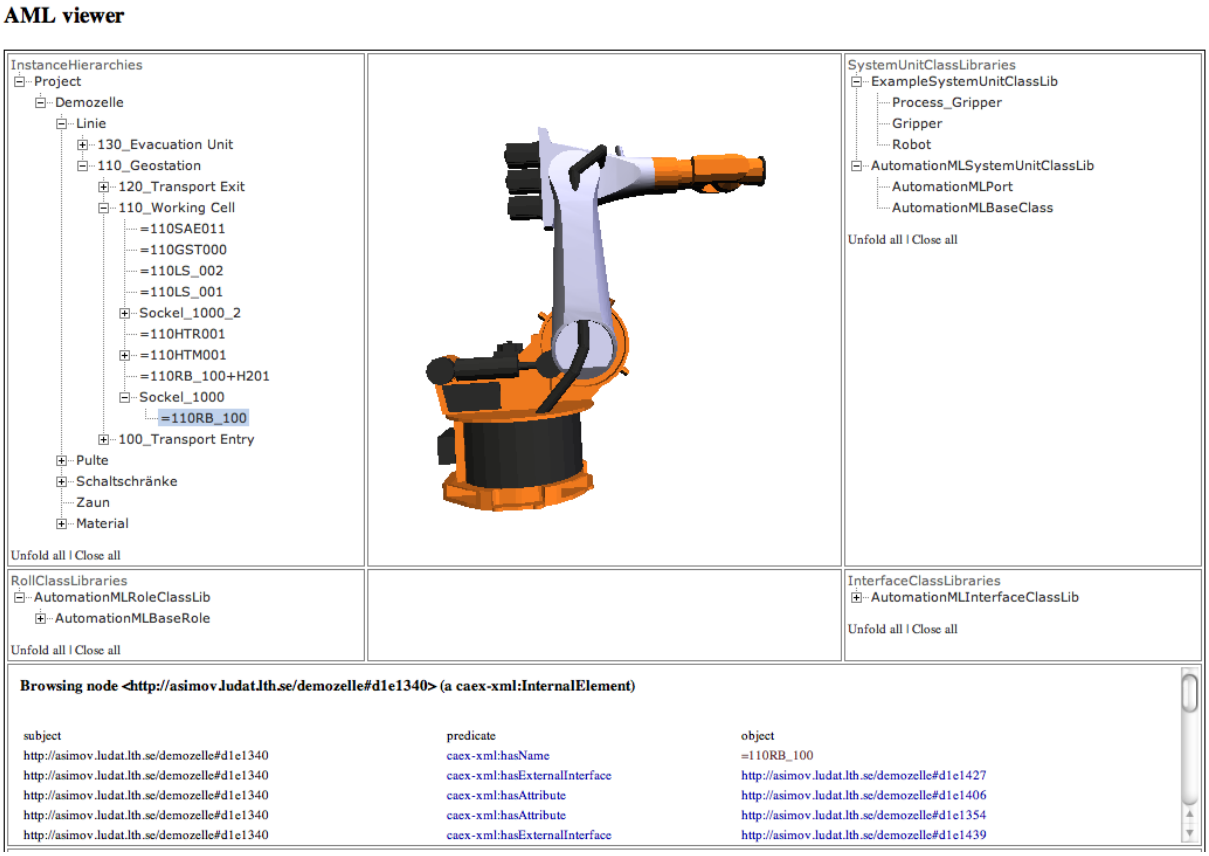

Figure 8: The KIF viewer showing the DemoZelle example available from `http://www.automationml.org/`.

and F. Breitenecker, editors, *Proceedings of 4th Mathmod Conference*, pages 1247–1256, Vienna, 2003.

[11] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[12] Roberto García and Òscar Celma. Semantic integration and retrieval of multimedia metadata. In *5th Knowledge Markup and Semantic Annotation Workshop, SemAnnot'05 CEUR Workshop Proceedings*, volume 185, pages 69–80, 2006.

[13] L. Hundt, A. Lüder, J. Peschke, and D. Weidemann. Description of logic data. Technical report, AutomationML consortium, April 2008.

[14] I. Miletic, M. Vujasinovic, N. Ivezic, and Z. Marjanovic. Enabling semantic mediation for business applications: Xml-rdf, rdf-xml and xsd-rdfs transformations. In *Enterprise Interoperability II*, pages 483–494. Springer, 2007.

[15] PLCopen. IEC 61131-3: Programmable controllers – part 3: Programming languages. Technical report, International Electrotechnical Commission, 2003.

[16] RDF. RDF/XML syntax specification. `http://www.w3.org/TR/rdf-syntax-grammar/`, February 2004.

[17] Leonard Richardson and Sam Ruby. *RESTful Web Service*. O'Reilly Media, 2007.

[18] Stefan Runde, Knut Güttel, and Alexander Fay. Transformation von CAEX-Anlagenplanungsdaten in OWL. Eine Anwendung von Technologien des Semantic Web in der Automatisierungstechnik. In *AUTOMATION 2009. Der Automationskongress in Deutschland*, pages 175–178. VDI Verlag, 2009.

[19] Simon Schenk. A SPARQL semantics based on Datalog. In *KI 2007: Advances in Artificial Intelligence*, pages 160–174. Springer, 2007.

[20] SPARQL. SPARQL protocol and RDF query language. `http://www.w3.org/TR/rdf-sparql-query/`, January 2008.

[21] Daniel Stenberg. curl and libcurl 7.19.7. curl-announce mailing list, 2009.

[22] Erik Wilde and Michael Hausenblas. RESTful SPARQL? You name it! Aligning SPARQL with REST and resource orientation. In *WEWST 2009*, November 9 2009.