Ontology-based Validation of Plant Models
Lisa Abele, Christoph Legat, Stephan Grimm
Siemens AG - Corporate Technology

Abstract—Improving efficiency in the plant construction process and providing valid resulting plant models requires a technology that validates planning and data exchange formats for plant engineering, such as CAEX (Computer Aided Engineering Exchange). Semantic Web technologies support validation mechanisms for querying and reasoning over domain models expressed in form of ontologies. In this paper, we present an approach to the automated validation of CAEX plant models by their transformation to ontologies and subsequent application of Semantic Web reasoning for validation purposes.

I. INTRODUCTION

The growing complexity of industrial plants and the wide variety of users that participate in the process of plant construction has made it increasingly difficult to exchange and to validate plant engineering models. The currently most recognized standard data exchange tool for plant engineering data that assists experts during the plant construction process is the XML-based data format Computer Aided Engineering Exchange (CAEX) [1] as part of the AutomationML (AML) language specification [2], which was specially developed to meet the requirements of the engineering domain.

During the plant construction process various experts with different backgrounds are involved, such as electrical engineers, mechanical engineers, etc. In the course of maintaining a CAEX model of the plant that integrates their views, various inconsistencies might be introduced, e.g. distinct elements accidentally get the same identifier, electrical connections are modeled incorrectly, etc. In order to work with a consistent model of the plant, such inconsistencies have to be identified and corrected. However, CAEX does not provide the built-in mechanisms to easily support automated validation of plant models. In particular, it does not come with a formal semantics that permits reasoning and querying over plant knowledge.

Semantic Web technologies, on the other hand, offer such mechanisms for querying and reasoning over domain models expressed in form of ontologies. They have further been identified as an enabler for future production systems by [3]. In particular, the Web Ontology Language (OWL), standardized by W3C, is equipped with a formal semantics, which accounts for a precise formulation of restrictions over domain models that can be validated using available reasoning and querying tools.

In this paper, we present an approach for the automated validation of CAEX plant models by means of their transformation into OWL ontologies and subsequent application of reasoning mechanisms. In particular, we provide a base ontology model that captures the essential CAEX notions in OWL fixing basic design decisions of ontological plant representation. Based on this, we specify a detailed mapping from CAEX models to OWL ontologies as a basis for automated transformation. Moreover, we demonstrate the use of reasoning and querying for some validation features along the proposed CAEX plant engineering process using Semantic Web reasoning and querying techniques.

II. PLANT ENGINEERING USING CAEX

In this section, the CAEX standard and its associated engineering process are introduced by means of a research facility recently engineered at Siemens. This demonstrator produces a configurable running text from small plastic disks using four conveyors each propelled by a separate motor (see Fig. 1). To ensure the correct interplay of the components, various requirements and parameters have to be considered in engineering [4], e.g. the maximum rotational motor speed.

IEC 62424 [5] defines CAEX as a meta model for the storage and exchange of engineering models. Below, its addressed topics are briefly summarized.

- Description of a concrete containment hierarchy of components (InstanceHierarchy, abbrev. IH), from top-level plant down to single components (InternalElements, IEs) with interfaces (ExternalInterfaces, EIFs) and relations (InternalLinks, ILs).
- Reusable SystemUnitClasses (SUCs) defining component types down to their respective technical realizations organized in vendor-specific product catalogues. Therein, hardware components are detailed by the vendor.
- Reusable role definitions for abstract descriptions of components (RoleClasses, RCs).
• Reusable Interface Classes (ICs) for specifying connection points of RCs, SUCs and the interface type of EIs.
• Attributes for describing characteristics of each previously introduced modeling element.

By definition, CAEX supports object-oriented modeling for all of these aspects. EIs depicting SUC instances are connected by ILs via EIs, which in turn are instances of ICs.

The engineering process using the CAEX standard consists of three major steps as depicted in Fig. 2. First, it is important to define the roles to be used in the respective domain. User-specific RCs are defined as specializations of standard RCs from the AML standard libraries. For creating a CAEX model of the demonstration facility, the basic RC Drive is refined as a specialized RC ConveyorDrive with constrained Attributes MaxRotationalSpeed and MaxEnergyConsumption and an IE defining a required ProfiNet communication feature. An IH representing a requirement-based containment view of the intended plant is created with IEs assigned to the concretized roles. Since various experts may be working on the engineering model concurrently, different IEs representing the same physical component might appear. To support the consistency of the model, such IEs must be identified.

As a second step, the defined RCs are used for selecting suitable components from vendor-specific product catalogues. Therein, available hardware components are defined and detailed by the vendor by means of SUCs. After manually selecting a suitable SUC, it is assigned to the previously defined IEs. Browsing the Siemens library, the SUC representing the electrical motor of type 1FK7 is selected for the IE representing the ConveyorDrive RC of the demonstration facility. Due to the manual SUC selection process and the variety of attributes to be considered, SUCs not fully meeting the role requirements might be assigned. Consequently, a mechanism for identifying such inconsistencies is beneficial.

In the third step, the IEs are connected by ILs, representing the plant-specific inter-component connections. Thus, the previously defined IH of the plant is completed. Here, the IH of the demonstration facility is completed by modeling, among others, the ILs for the communication connections of the motor based on its ProfiNet port represented as EI. Due to the complexity of a plant model, incorrectly connected IEs might occur. To identify such inconsistencies as early as possible, ILs need to be checked for properly connected EIs.

III. PLANT REPRESENTATION IN ONTOLOGIES

In this section, we propose a base ontology model for capturing the basic notions of CAEX together with a transformation process for mapping CAEX plant models to ontologies, after we briefly introduce the Web Ontology Language (OWL) used as a representation framework.

A. Web Ontology Language (OWL)

The Web Ontology Language OWL [6] is a W3C standard for representing ontologies, the main building blocks for expressing knowledge in the Semantic Web. It can be used to express taxonomies of classes and relationships between conceptual notions, such as the various components of a plant, and is thus apt for representing plant knowledge.

The basic elements of OWL are classes (like Motor1FK7), individuals (like the specific motor m1) as instances of classes, and properties between individuals (like wiredTo). Statements in OWL, using the class, individual and property vocabulary are called axioms. We consider the most recent and prominent language variant OWL DL, which distinguishes terminological knowledge at class level (e.g. axioms stating a subclass relationship) from assertional knowledge at instance level (e.g. axioms asserting an individual as an instance of a class). A specialty of OWL is to support logic-inspired language constructors for forming complex classes. An example of an assertional axiom using a complex class expression is the following: ClassAssertion(ObjectIntersectionOf(Motor Object SomeValuesFrom(hasManufacturer "Siemens") 1FK7)), which states that the individual 1FK7 is an instance of the complex class made of all motors that have “Siemens” as manufacturer.

OWL has support for reasoning and querying, which allows for the validation of and for the retrieval of implicit knowledge from ontologies. We will use these mechanisms for validating plant models obtained from their original CAEX descriptions. For querying, we build on the SPARQL [7] query language under an OWL-compliant entailment regime, as described in [8].

B. Base OWL Model

Our transformation from CAEX to OWL builds on a base ontology, denoted by $O_{base}$, which captures some basic design decisions of representing CAEX plant models in OWL ontologies. Figure 3 shows the OWL representation of the major CAEX notions in $O_{base}$ as an UML class diagram.

Following a suggestion in [9], SUCs, as well as RCs and ICs, are represented at the terminological level in terms of OWL classes. The IEs of a given IH are then represented at the assertional level as instances of the basic class $b$ SystemUnit.
Containment between IEs according to the IH is expressed by means of the OWL property \texttt{b:partOf}.

There are two possibilities to connect SUCs with RCs. If a SUC fulfills all requirements defined in a RC it is connected via the property \texttt{supportsRoleRequirementsOf} to its RC and plays exactly the specified RC in the industrial plant, e.g. the “Motor 1FK7” plays the role of a “Conveyor Drive”. All other RCs the SUC can potentially support are connected by \texttt{supportsRole}, e.g. the “Motor 1FK7” can play the role of a “Mixer Drive” or “Pump Drive”.

The requirements to be fulfilled by a SUC are specified in \texttt{b:Interface} and \texttt{b:Attribute} classes connected to the RC whose requirements it is said to support. The RC may require attributes expressed by the property \texttt{b:requiresInterface}. In a similar way, an RC can require the respective SUC to support ICS, which is represented by means of the property \texttt{b:requiresAttribute}.

The properties \texttt{b:hasAttribute} and \texttt{b:providesInterface} are used to specify all attributes and ICS attached to a SUC. The property \texttt{b:isLinkedTo} is meant to express \texttt{IL}s between the \texttt{IE}s related to ICS. Thus, we represent ICS in terms of the class \texttt{Interface} and \texttt{IE}s in terms of instances thereof.

C. Mapping CAEX to OWL

We introduce a transformation process in which a custom plant model described in a specific CAEX document is mapped to a respective OWL ontology \(O_{\text{plant}}\) that represents this plant model in alignment with the basic notions introduced in \(O_{\text{base}}\). The resulting ontology \(O_{\text{plant}}\) is based on the vocabulary from \(O_{\text{base}}\) by means of \texttt{owl:imports}. In fact, standard libraries, such as vendor-specific component catalogues used by the original CAEX model, can be transformed separately in our approach to result in separate ontologies that have different namespaces and are linked via \texttt{owl:imports} as well.

Table I specifies our proposed transformation process in detail by indicating the mapping of CAEX elements on the left-hand side to corresponding OWL elements on the right-hand side in terms of syntax, semantics and nature of the respective language element.\(^1\) The top part of Table I describes the mapping of the basic elements of a CAEX model, whereas the main part specifies how relations between basic CAEX elements, such as XML-nesting or the use of reference attributes, are mapped to OWL. Variables written in italic are used in places where text fragments from the original CAEX model are reused in OWL syntax expressions. In particular, the names of SUCs, RCs and ICS are reused for naming their respective class entity in \(O_{\text{plant}}\), assuming that they are consistently and uniquely named in the CAEX model. For IEs, however, no such assumption is made and new identifiers for their corresponding OWL individuals are created during transformation according to a number scheme, such as simply counting the occurrences of name/id pairs of IEs. In this sense, \(ie\#\) stands for the individual that corresponds to the \(IE\) in the context of the currently considered CAEX IE XML tag.

A plant’s containment hierarchy encoded in the nesting of CAEX internal elements, for example, is mapped to OWL object property assertions between the respective named individuals using the basic property \texttt{b:partOf}. On the other hand, attributes of SUCs, IEs, etc. are instantiated anonymously by existential restrictions at the terminological level: instead of assigning a named individual as an attribute, the respective SUC is stated to have an attribute with the specified name and value in a \texttt{owl:subClassOf} statement. Furthermore, we make use of \texttt{owl:sameAs} statements between \(IE\) individuals to represent the notion of CAEX mirror elements, i.e. IEs that represent the same plant element.

IV. REASONING SUPPORT FOR PLANT ENGINEERING

In the following, we will exemplify three consistency checks along the CAEX process that we have implemented with Semantic Web technologies. An ontology of the application scenario as defined in section II serves as our example. First, we support the operator in identifying identical IEs. Further, we validate the RC requirements of all IEs and then, we validate the correctness of IILs.

A. Select identical Internal Elements

CAEX allows several plant operators with different expertise to extend a plant hierarchy with additional IEs during the first phase of the CAEX engineering process. It is possible that two distinct elements accidentally have the same Id, name and RC. According to the standard IEC 62424 two IEs are identical if their names, Ids and RCs are equivalent. Based on this criteria, our aim is to select all IEs with a SPARQL query to support the operator in deciding if these IEs were meant to be identical or if they accidentally have the same identifiers. An exemplary ontology with two identical IEs is shown in Figure 4. IEs of two different SUCs “Motor1FK7” and “Motor1PH8” have identical Ids and names and they support the requirements of the same RC “ConveyorDrive”. Our SPARQL query applied on the exemplary ontology identifies the individuals “ie1” and “ie2”. If these two IEs were meant to be different, the operator can change their Id, name or RC. The SPARQL query expressed in the OWL functional-style syntax has the following structure:

\footnote{We use OWL functional style syntax (www.w3.org/TR/owl2-syntax/).}
SELECT ( ? i e 1 ? name ? i d ? i e 2 ) WHERE {
AnnotationAssertion ( rdf:label ? i e 1 ? name )
DataPropertyAssertion ( b:hasID ? i e 1 ? i d )
DataPropertyAssertion ( b:hasID ? i e 2 ? i d )
ClassAssertion ( ObjectSomeValuesFrom ( b:supRoleReqOf ? role ) ? i e 1 )
ClassAssertion ( ObjectSomeValuesFrom ( b:supRoleReqOf ? role ) ? i e 2 )
FILTER ( ? i e 1 != ? i e 2 )
}

B. Validate Role requirements of Internal Elements

In the second step of the CAEX engineering process, appropriate SUCs for the IEs based on the requirements defined in the according RCs are manually selected. An issue of this selection task is that the operator is often confronted with an enormous list of requirements. It might thus happen that requirements of an IE as defined by its RC are not entirely fulfilled by its SUC. To support the operator, we validate RC requirements by checking the attributes of all RCs and SUCs associated to an IE. We define a SPARQL query to identify all IEs where the attributes do not match. We use SPARQL queries under OWL entailment regime expressed in OWL functional-style syntax:

ClassAssertion ( ObjectSomeValuesFrom ( b:prescribesAttribute ObjectIntersectionOf ( DataHasValue ( b:hasValue ? valR ) DataHasValue ( b:hasName ? a ))) SubClassOf(? role ObjectSomeValuesFrom ( b:hasIntAttribute DataHasValue ( b:hasValue ? valIE ) DataHasValue ( b:hasName ? a )))
FILTER ( ? valR != ? valIE )

In Figure 5, we demonstrate an example. We validate the attributes of IE Motor1FK7 which is an instance of Motor1FK7 and supports the role requirements of the RC ConveyorDrive. This RC requires an attribute maxEnergyCons with a value 500 W and an attribute maxRotSpeed with a value of 1000 rpm. Motor1FK7 supports only the required value of attribute maxEnergyCons, a conflicting attribute is maxRotSpeed since its value 1080 is too high for the ConveyorDrive (Note: a higher speed of the conveyor belt may cause serious damages).

A similar query can be defined to validate the attributes of EIs required by RCs.

C. Validate correctness of Internal Links

User-defined libraries play an important role for the development of exchangeable plant models in CAEX. These libraries, e.g. AML BaseLibs, provide an extended vocabulary with additional semantics not provided by CAEX. Such semantics offers a possibility of being validated.

The standard AMLBaseInterfaceLib for example defines Signal Interfaces (SIs) with an attribute “Direction”, with possible values “In”, “Out” or “InOut”. According to the AML standard, SIs with the direction “In” or “Out” can only be connected to SIs with opposite direction or “InOut”. SIs with the direction “InOut” can be connected to SIs of arbitrary direction. In order to validate correct wiring, this semantics needs to be defined formally.

To support operators that use the SI attributes of the AMLBaseInterfaceLib, we identify all SIs with direction “In” or “Out” which are connected to an IC of the same direction. The ontology $O_3$ has the following structure:

$$O_3 = \text{SubClassOf}(\text{SignalInterface}) \text{ObjectIntersectionOf}(\text{SignalInterface} \text{ObjectSomeValuesFrom}(b:hasAttribute \text{ObjectIntersectionOf}(\text{DataHasValue}(b:hasValue "In") \text{DataHasValue}(b:hasName "Type")))), \text{SubClassOf}(\text{SignalInterfaceOut}) \text{ObjectIntersectionOf}(\text{SignalInterface} \text{ObjectSomeValuesFrom}(b:hasAttribute \text{ObjectIntersectionOf}(\text{DataHasValue}(b:hasValue "Out") \text{DataHasValue}(b:hasName "Type"))))$$

EquivalentClass(\text{MiswiredSignalInterface} \text{ObjectIntersectionOf}(\text{ObjectIntersectionOf}(\text{ObjectSomeValuesFrom}(b:isLinkedTo \text{SignalInterfaceIn}) \text{SignalInterfaceOut}) \text{ObjectIntersectionOf}(\text{ObjectSomeValuesFrom}(b:isLinkedTo \text{SignalInterfaceOut}) \text{SignalInterfaceIn})))

Based on $O_3$, individuals, which are inferred to be instances of the MiswiredSignalInterface class, can be identified as incorrectly wired. As depicted in Fig. 6, the EIs
are classified based on their direction attribute to the SignalInterfaceIn or SignalInterfaceOut, respectively. Since the EI ei1, which has the direction “In”, is linked to EI ei2, which has the same direction, ei1 is additionally identified as MiswiredSignalInterface.

V. RELATED WORK

In recent years, there has been an increasing amount of literature on semantic ontologies in manufacturing as stated in [10]. For example, the meaning of semantic technologies for integration of engineering tools (e.g. MCAD, ECAD, etc.) is highlighted in [11]. In [12] a maintenance and process ontology in operation & maintenance information management is applied, but there is no binding to existing CAE standards or tools and no discussion on how to embed the ontology into the engineering process. An approach to overcome the limits of XML encoded device descriptions by employing Semantic Web technologies is presented in [13]. We could use the resulting device description ontology to extend our ontology by storing more detailed knowledge about the devices. Another example of successful deployment of Semantic Modelling in manufacturing is presented in [14]. Their proposed ontology could be enhanced by our proposed ontology as well as our validation features. [9] defines transformation rules for mapping CAEX into OWL ontologies. In contrast to our approach, this research does not propose additional query or validation features. A transformation from the COMOS data model to OWL is described in [15], but the authors focus on runtime application rather than on engineering. These approaches point out the multiple advantages of OWL-DL compared to CAEX or COMOS data models.

Several approaches investigated in validating and querying CAEX files with usage of semantic ontologies. An approach proposed by [16] focusses on converting CAEX files into RDF triples to expose them via a SPARQL endpoint in robotized production. It builds on rules defined by [9] to convert CAEX documents into OWL. This approach presents application-specific mappings and offers only querying of the CAEX model, but no consistency checks. A validation approach for CAEX without usage of semantic ontologies was published by [17]. Their use case is close to ours since they detect incomplete or wrong designs in early stages of the engineering development cycle. However, they define a Markup Language based on XML which impedes extension and enhancement of models in general as found in [18]. The CAEX tool suite as proposed in [19] offers some mechanisms similar to ours, e.g. the elimination of redundant information, but manual validations have to be executed by the user. Available CAEX editors, e.g. the AutomationML editor, support our features partially, but they are based on a formal model which does not allow to infer over semantic models, e.g. by using a taxonomy of interfaces.

VI. CONCLUSION

We have presented an approach to the automated validation of plant models that are originally expressed in the CAEX exchange format based on Semantic Web technology. To this end, we have described a base model for ontological plant representation in the ontology language OWL and a detailed transformation of CAEX elements to OWL constructs. Moreover, we have exemplified the automated validation of plant models based on various validation features that support different phases of the CAEX plant engineering process by making use of standard reasoning and querying techniques.

This work constitutes a starting point for a comprehensive investigation of the various situations that may occur in plant modeling on a semantic level, and for systematically collecting appropriate validation features to cope with these situations. By offering such a toolbox of features, the interdisciplinary cooperation of plant engineers during the plant modeling process can be considerably improved.

ACKNOWLEDGMENT

This research was funded in part by the German Federal Ministry of Education and Research under grant number 01IA11001.

REFERENCES

<table>
<thead>
<tr>
<th>Element</th>
<th>Syntax</th>
<th>Semantics</th>
<th>Element</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>basic elements</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUC</td>
<td><code>&lt;SystemUnitClass Name=&quot;SU&quot;&gt;</code></td>
<td>class of System Units</td>
<td>owl:Class SU</td>
<td>SubClassOf(SU SystemUnit)</td>
</tr>
<tr>
<td>RC</td>
<td><code>&lt;RoleClass Name=&quot;R&quot;&gt;</code></td>
<td>class of Roles</td>
<td>owl:Class R</td>
<td>SubClassOf(R Role)</td>
</tr>
<tr>
<td>IC</td>
<td><code>&lt;InterfaceClass Name=&quot;I&quot;&gt;</code></td>
<td>class of Interfaces</td>
<td>owl:Class I</td>
<td>SubClassOf(I Interface)</td>
</tr>
<tr>
<td>IE</td>
<td><code>&lt;InternalElement Name=&quot;ie&quot; ID=&quot;id&quot;&gt;</code></td>
<td>System Unit instance</td>
<td>owl:NamedIndividual ie#</td>
<td>ClassAssertion(SystemUnit ie#) AnnotationAssertion(rdfs:label ie# &quot;ie&quot;) DataPropertyAssertion(b:hasID ie# &quot;id&quot;)</td>
</tr>
<tr>
<td>EI</td>
<td><code>&lt;ExternalInterface Name=&quot;ei&quot; ID=&quot;id&quot;&gt;</code></td>
<td>Interface class instance</td>
<td>owl:NamedIndividual ei#</td>
<td>ClassAssertion(SystemUnit ei#) AnnotationAssertion(rdfs:label ei# &quot;ei&quot;) DataPropertyAssertion(b:hasID ei# &quot;id&quot;)</td>
</tr>
<tr>
<td><strong>relations</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nesting of IEs</td>
<td><code>&lt;InternalElement Name=&quot;ie&quot;&gt;</code></td>
<td>part of relation</td>
<td>property assertion axiom</td>
<td>ObjectPropertyAssertion(b:partOf ie# ei#)</td>
</tr>
<tr>
<td>nesting of IE &amp; Attribute</td>
<td><code>&lt;Attribute Name=&quot;a&quot;&gt;</code></td>
<td>attribute instantiation</td>
<td>class assertion axiom</td>
<td>ClassAssertion(ObjectSomeValuesFrom(b:hasAttribute ObjectIntersectionOf(DataHasValue(b:hasName &quot;a&quot;) DataHasValue(b:hasValue &quot;v&quot;))) ie#)</td>
</tr>
<tr>
<td>nesting of IE &amp; EI</td>
<td><code>&lt;ExternalInterface Name=&quot;ei&quot;&gt;</code></td>
<td>Internal Element provides interface</td>
<td>property assertion axiom</td>
<td>ObjectPropertyAssertion(b:providesInterface ei# ei) ClassAssertion(I ei)</td>
</tr>
<tr>
<td>relation IE &amp; SUC</td>
<td><code>&lt;InternalElement Name=&quot;ie&quot;&gt;</code></td>
<td>copy of</td>
<td>class assertion axiom</td>
<td>ClassAssertion(SU ie#)</td>
</tr>
<tr>
<td>nesting of SUC &amp; EI</td>
<td><code>&lt;SystemUnitClass Name=&quot;SU&quot;&gt;</code></td>
<td>System Unit provides interface</td>
<td>class inclusion axiom</td>
<td>SubClassOf(SU ObjectSomeValuesFrom(b:providesInterface I)))</td>
</tr>
<tr>
<td>nesting of SUC &amp; Attribute</td>
<td><code>&lt;SystemUnitClass Name=&quot;SU&quot;&gt;</code></td>
<td>System Unit has attribute</td>
<td>class inclusion axiom</td>
<td>SubClassOf(SU ObjectSomeValuesFrom(b:hasAttribute ObjectIntersectionOf(DataHasValue(b:hasName &quot;a&quot;) DataHasValue(b:hasValue &quot;v&quot;))) su#)</td>
</tr>
<tr>
<td>nesting of RC &amp; EI</td>
<td><code>&lt;RoleClass Name=&quot;R&quot;&gt;</code></td>
<td>Role requires interface</td>
<td>class inclusion axiom</td>
<td>SubClassOf(R ObjectSomeValuesFrom(b:requiresInterface EI)) SubClassOf(EI I)</td>
</tr>
<tr>
<td>nesting of IC &amp; Attribute</td>
<td><code>&lt;InterfaceClass Name=&quot;I&quot;&gt;</code></td>
<td>interface attribute</td>
<td>class inclusion axiom</td>
<td>SubClassOf(I ObjectSomeValuesFrom(b:hasAttribute ObjectIntersectionOf(DataHasValue(b:hasName &quot;a&quot;) DataHasValue(b:hasValue &quot;v&quot;))) i#)</td>
</tr>
<tr>
<td>relation SUC &amp; RC</td>
<td><code>&lt;SystemUnitClass Name=&quot;SU&quot;&gt;</code></td>
<td>supports role requirements</td>
<td>class inclusion axiom</td>
<td>SubClassOf(SU ObjectIntersectionOf(ObjectSomeValuesFrom(b:supportsRoleRequirementsOf R) R))</td>
</tr>
<tr>
<td>relation IE &amp; RC</td>
<td><code>&lt;InternalElement Name=&quot;ie&quot;&gt;</code></td>
<td>supports role</td>
<td>class assertion axiom</td>
<td>ClassAssertion(ObjectSomeValuesFrom(b:supportsRoleRequirementsOf R) ie#)</td>
</tr>
<tr>
<td>relation IE &amp; RC</td>
<td><code>&lt;RoleClass Name=&quot;R&quot;&gt;</code></td>
<td>supports role</td>
<td>class assertion axiom</td>
<td>ClassAssertion(ObjectSomeValuesFrom(b:supportsRole R) i#)</td>
</tr>
<tr>
<td>identical IEs</td>
<td><code>&lt;InternalElement Name=&quot;ie&quot;&gt;</code></td>
<td>mirror element</td>
<td>individual equality axiom</td>
<td>SameIndividual(ie# ei#)</td>
</tr>
<tr>
<td>inherited SUCs</td>
<td><code>&lt;SystemUnitClass Name=&quot;SU&quot;&gt;</code></td>
<td>inheritance</td>
<td>class inclusion axiom</td>
<td>SubClassOf(SUC1 SUC2)</td>
</tr>
<tr>
<td>internal link between Els</td>
<td><code>&lt;InternalLinkName=&quot;link&quot;&gt;</code></td>
<td>connection</td>
<td>property assertion axiom</td>
<td>ObjectPropertyAssertion(b:isLinkedTo ei# ei2)</td>
</tr>
</tbody>
</table>

**TABLE I:** Mapping between CAEX and OWL, SUC = SystemUnitClass, RC = RoleClass, IC = InterfaceClass, IE = InternalElement, EI = ExternalInterface, ie# = identifier for IEs