Internship ZEnMo Simulations

Energy System Description Language integration in AnyLogic models

Author:
Maarten Bastiaansen

Master Sustainable Energy Technology
Technology, Innovation & Society research group

Supervisors:
Floor Alkemade (TU/e)
Naud Loomans (ZEnMo Simulations)
Peter Hogeveen (ZEnMo Simulations)

Eindhoven, July 13, 2022

Preface

This report is a documentation of the findings of a research project during my internship at ZEnMo Simulations from May 2022 till July 2022. The internship is part of my master program Sustainable Energy Technology at the Eindhoven University of Technology.

Contents

1 Research context

The energy transition is one of the largest challenges of the present-day. It requires development and integration of a wide range of renewable energy technologies in combination with various social, economic and legal aspects of these technologies. To tackle renewable energy related challenges, policy makers should turn to advanced energy computer models and simulation tools for consultation and useful insights. However, such models and tools are independently developed by a wide range of parties in the field. This leads to overall dissimilarity, incomparability and incompatibility between models. A standardized method can get everyone on the same wavelength, thus increasing efficiency and usability, while decreasing complexity and costs. The Dutch research organization *Nederlandse Organisatie voor toegepast-natuurwetenschappelijk onderzoek* (TNO) has developed an universal open source programming language including support tools that allow for the usage of an uniform modelling language across all different kind of organizations that operate in the energy system modelling field. This tool is the so-called Energy System Description Language (ESDL) [TNO, 2022g].

1.1 Energy System Description Language

As the name suggests, ESDL is a programming language. Just like with any exchange of information, communication is much more simplified if all parties involved speak exactly the same language. This is precisely what ESDL is trying to achieve for the modelling of energy systems. Uniformly capturing an abstraction of a physical energy system which multiple different computational models can use is at the core of what ESDL is all about. It is designed to describe the components of an energy system as well as the relations between these components in a systematic format [TopSectorEnergie, 2022].

ESDL facilitates communication between different energy models. Simulation results from different models can be combined and analysed through ESDL. Changing assumptions, input data or insights in the energy model can therefore be more efficiently be handled through this uniform language. Besides this, energy models focusing on the same domain but from different parties currently often result in completely different results on the same research/policy question [TNO, 2022i]. By publicising assumptions (such as costs, efficiencies and energy profiles) in a standard format, transparency and comparability to other models is increased.

The main advantages of ESDL can be summarized as follows [TNO, 2022i]:

- Universal programming language for energy systems that is modelling software independent.
- Combining social, financial, legal and technical characteristics in one uniform language for energy system models.
- Hierarchical structurization of components and energy assets in energy system models.
- The possibility of systematic communication between multiple energy system models.
- Accessibility to other existing energy system models for exploring functionalities.
- Increasing transparency of energy system models.
- Documenting and standardizing assumptions and data in energy system models, such as CO_2 emissions, cost (developments) and technological fact sheets of assets.

According to TNO, approximately 40 actors in the Netherlands make use of ESDL in the simulation of energy systems, which is expected to increase significantly. The focus of ESDL is on the Dutch sector in energy consultancy and simulation, however, TNO has ambitions to use it as an international standard.

ESDL is set up in such a structured way that it combines three types of information of an energy system [TNO, 2022c]. The first type is the registration of the energy system. This type provides geographical data that contains information such as building characteristics, energy producer/consumer assets and characteristics of transportation networks. The second information type can be classified as the potential of the energy system; information about where wind turbines can be installed, where geothermal energy can be found and which areas are suitable for heat networks. The final type of energy system information is that of the usage of the energy system over time. It describes the dynamic behaviour over time. These are typically energy profiles or measurements, such as the energy consumption of a building or the solar energy production of a PV panel. Usually these three types of information are provided by separate sources, ESDL combines these in one common language.

Based on the three types of information, energy assets in ESDL are ordered into five types of capabilities; production, consumption, storage, transport and conversion of energy. This is shown in figure 1. Each capability has its own standardized characteristics. Categorizing in such a way allows for making an abstraction of models by only modelling characteristics that are relevant for that capability. This in turn allows for relative simple aggregation of the components in the energy system on any level (for instance on house, neighborhood or city level) [TNO, 2022a].

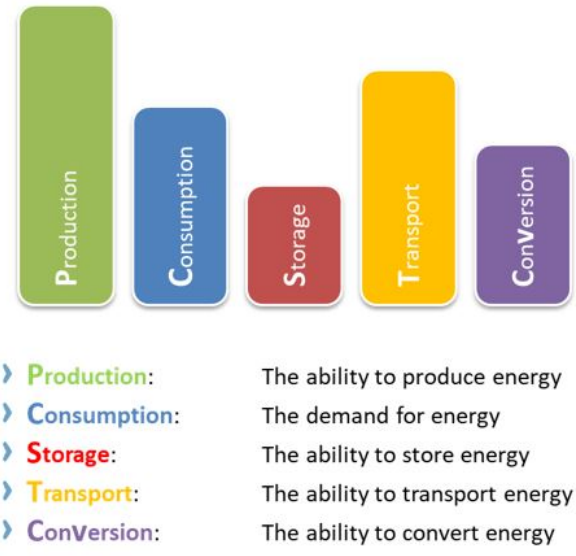

Figure 1: Overview of the five capabilities [TNO, 2022a]

1.2 Energy system modelling in AnyLogic

The ESDL ontology is mainly of high value in a preliminary and an exploratory phase of extensive energy systems development, while it is of less value when zooming in on a small number (or even one specific) component in the energy system. Gaining insights in this all-embracing system integration is precisely what ZEnMo Simulations core business is. ZEnMo Simulations focus on agent-based modelling of energy systems by using the Java-based AnyLogic programming tool. Agent based modelling of energy systems allows for a number of extra features as compared to other type of energy system modelling. Human technology interaction, simulating through time and scenario analysis are all features that are easy to incorporate in agent based models. AnyLogic allows for simulation of an energy system through time in an user-friendly interface. By developing complex energy models with an easy-to-use user interface, ZEnMo Simulations is able to provide stakeholders in an energy system with insights into the working principles of it. This could help these stakeholders in forming proper policies or making the right investment in a renewable energy system.

1.3 Research goal

This research focuses on exploring the possibilities and opportunities of integrating ESDL characteristics and data files in energy systems designed in AnyLogic. The ESDL characteristics that are explored for integration are the universality, compatibility and standardization and structurization of energy models. In this research the goal is to design a process that embeds these characteristics from an ESDL file in an AnyLogic model in such a way that is most relevant for ZEnMo Simulations.

2 Research approach

For achieving the research goal of obtaining a functional and advantageous synthesis between ESDL and AnyLogic, an exploration of the possibilities and opportunities in both programming tools is required. By first exploring the options of the existing tools and functions separately, it is possible to get a sense of the potential of the combination of the two. As ZEnMo Simulations has no experience with ESDL, a deep dive in the topic will be necessary.

Based on the adequate examination of ESDL, energy systems in the ESDL ontology will be created. This will be done in the ESDL-supported programs Eclipse and the ESDL MapEditor. These systems can be saved as an ESDL file. Subsequently, these ESDL systems can be loaded in in other modelling tools (such as AnyLogic) if ESDL importers are designed. For the integration of ESDL in AnyLogic, a Java-based importer of ESDL files in AnyLogic will be designed. This importer will open, read and process the ESDL files and all the relevant data in it. This ESDL imported data will then be used to construct the energy system in AnyLogic and to expand on the functionalities. In figure 2 an overview is given of these steps.

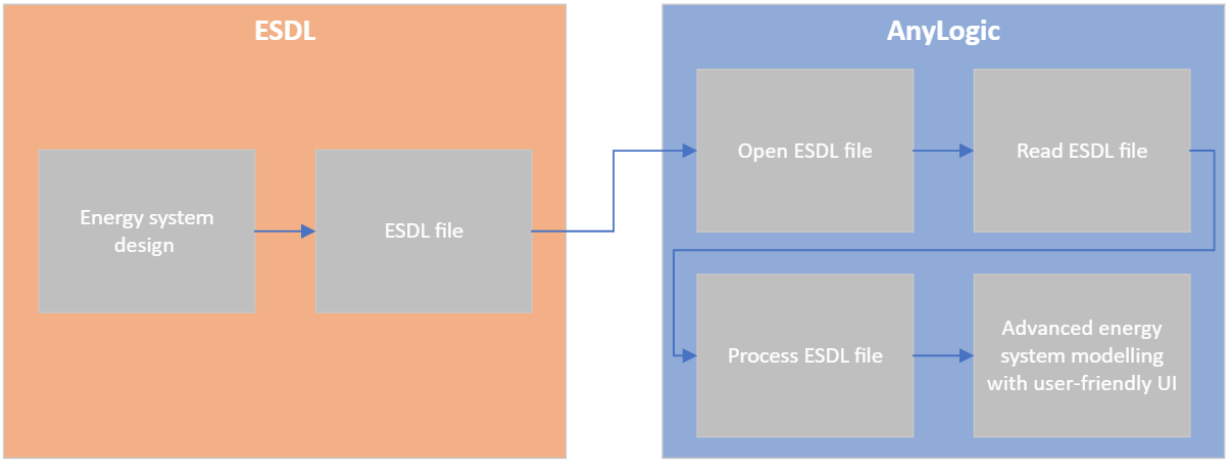

Figure 2: Overview of the research methodology

The functioning of the constructed importer can be tested in a specific energy system case. In this research the case of a heat network in the industrial area of Apeldoorn-Noord has been chosen. This area and energy system have been chosen because it is also explored in other running projects of ZEnMo Simulations. The added value of this importer can thus be immediately inspected and potentially even be integrated in this running project. The components of the heat network that will be simulated will be part of a very simple heat network, as the point of this case is to only show the functionalities of the ESDL importer in AnyLogic. With the functioning of this importer the step towards more complex systems can easily be made by ZEnMo Simulations due to the fact that more complex models are already built by the company.

3 Research results

In this chapter the results of this research are shown. First the workings of designing energy systems in ESDL are presented in section 3.1. Following up, the results for the integration of ESDL in AnyLogic are presented in section 3.2. This section includes the Java-based importer itself as well as the application to the case of the industrial area Apeldoorn-Noord.

3.1 Energy system design in ESDL

Designing an energy system in the ESDL ontology can be done in a number of ways depending on the desired features. The most straight-forward way is either in the program Eclipse or in the program ESDL MapEditor. This section discusses both ways as well as the resulting ESDL files and the structure of such files.

3.1.1 Eclipse

In the Java-based development program Eclipse it is possible to design and describe an energy system in the ESDL ontology [TNO, 2022b]. This program can be of great interest when designing generic energy systems, as no specific geographical attributes are assigned in this way of modelling in ESDL. Once such a generic energy system is designed it can be used for multiple different cases, without the necessity to alter a lot to the design due to the structured and all-embracing setup of ESDL. An example of such an energy system can be seen in figure 3. This figure shows a simple conventional energy system in neighborhood A on the left side, while on the right side a more complex decentralized energy system in neighborhood B is shown. Each component is categorised in one of the five capabilities (production, consumption, storage, transport and conversion) and colored accordingly. Besides this, each component can be connected to other components and has its own attributes. These attributes can be related to for instance the costs, state of matter, energy content, emissions, efficiency etc.

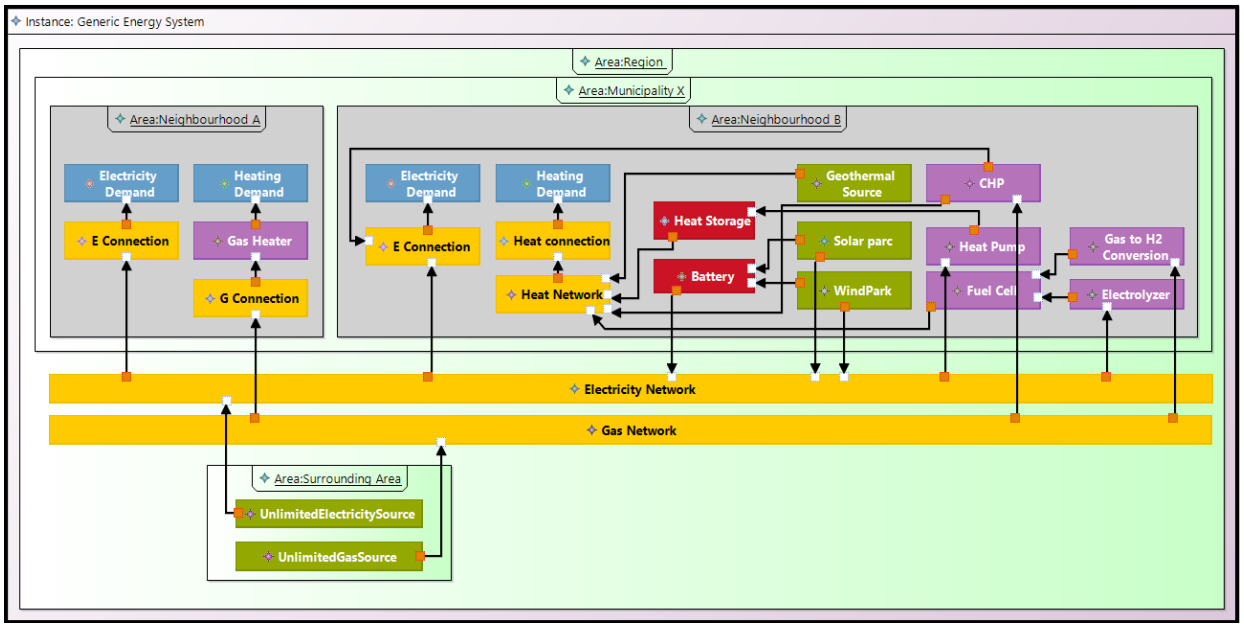

Figure 3: Graphical representation of an energy system as an ESDL file in Eclipse

The same system can be presented as an hierarchical tree in Eclipse as shown in figure 4. This also provides information regarding the overall energy system, such as the types of energy carriers, their quantity and units and the parties/stakeholders involved in this energy system.

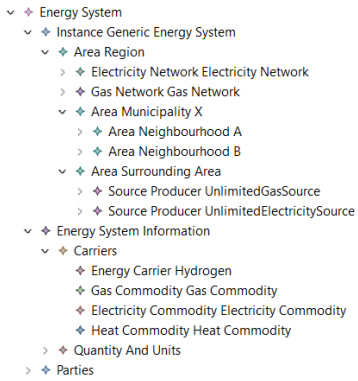

Figure 4: ESDL hierarchical tree in Eclipse

3.1.2 ESDL MapEditor

Another way for designing an energy system in ESDL is by constructing it in the web-based ESDL MapEditor [TNO, 2022d]. It has been developed to create an ESDL based energy system by simply dragging and dropping energy system components on a map. This tool brings a number of added features which can assist in the design process for more specific and detailed energy systems. The main added feature are the geographical and physical properties; coordinates, visualization of building shapes and BAG-data (building types, building year and floor area) are all integrated in the MapEditor. Besides these properties, the structured way in which ESDL handles complete buildings (with possibly a number of buildingunits per building) and energy assets in each building(unit) makes this the ideal setup in the first design steps of an energy system in an area.

Figure 5 shows an example of an energy system designed in the ESDL MapEditor, in this case for the industrial area Apeldoorn-Noord. The colored buildings are all buildings that are loaded in from BAG-databases, while the circular shaped icons are components of the energy system in this neighborhood.

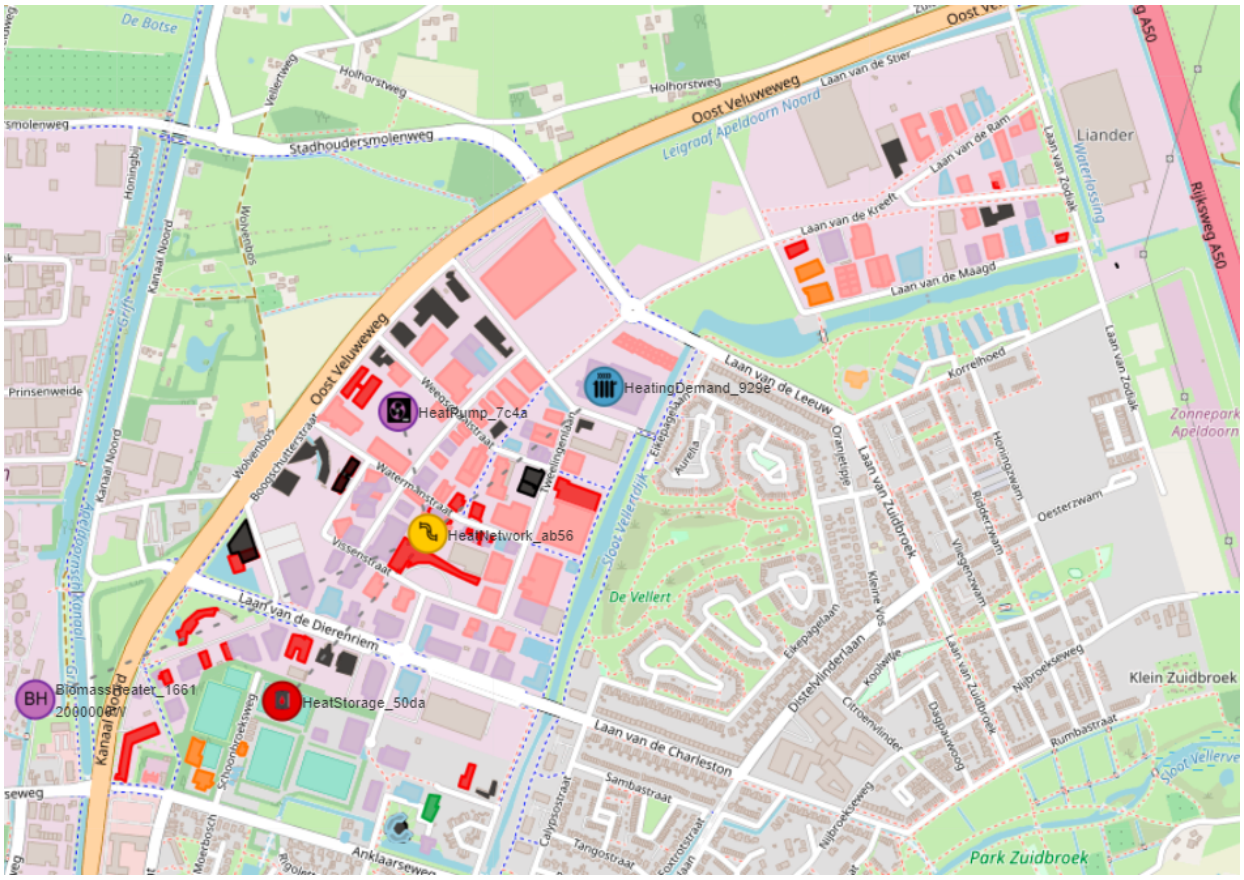

Figure 5: Industrial area in Apeldoorn-Noord with basic heat network assets designed in the ESDL MapEditor

3.1.3 ESSIM

The Energy System Simulator (ESSIM) [TNO, 2022f] is an online tool that allows for analysing energy flows of a designed energy system in the ESDL MapEditor. ESSIM mainly focuses on energy balancing and emission calculations. While these are important key performance indicators, they do not meet all the requirements and needs ZEnMo Simulations has for their models, such as simulating through time and user friendly interfaces. Moreover, it has very limited options to include flexibility and different energy management systems. Besides that, these topics can also be analysed in models in AnyLogic. On top of earlier mentioned reasons, it is not advised to use the ESSIM tool but to create an importer from any ESDL system design tool (such as Eclipse or the MapEditor) and analyse it fully in AnyLogic.

3.1.4 ESDL file

The ESDL files are typically structured in a very hierarchical way, as can be seen in the figure 6. Such a file is in essence what ESDL is all about. Generally speaking, an energy system in ESDL consists of an instance in which area(s) exist. Each area can have buildings and assets, each with their own specific characteristics.

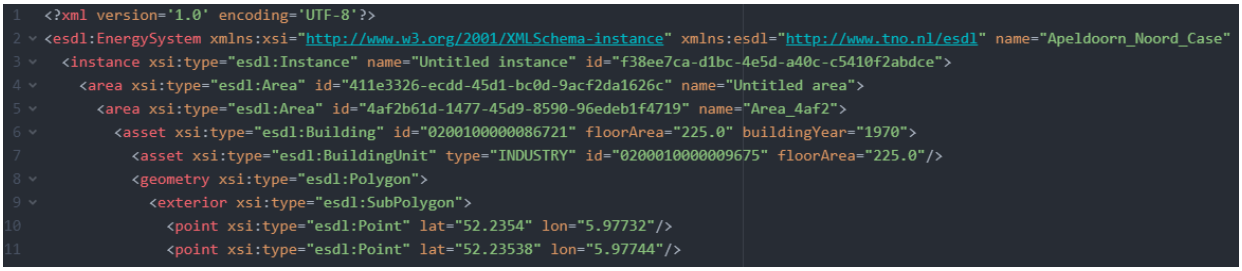

Figure 6: Part of the ESDL file of the Apeldoorn-Noord case

This hierarchical structure is documented in the *ESDL Model Reference documentation* [TNO, 2022e]. This documentation has been consulted a lot in this research for creating the ESDL importer in AnyLogic. It is very useful for exploring the attributes of an asset and their properties. Figure 7 shows an example of this documentation and therefore how ESDL is structured in its core. In this example the documentation for an heat network is shown. In the figure a number of attributes (properties of the energy asset) are shown, such as capacity, efficiency and ID number. It also shows as what kind of 'type' this data is stored, for instance as an integer, string or double.

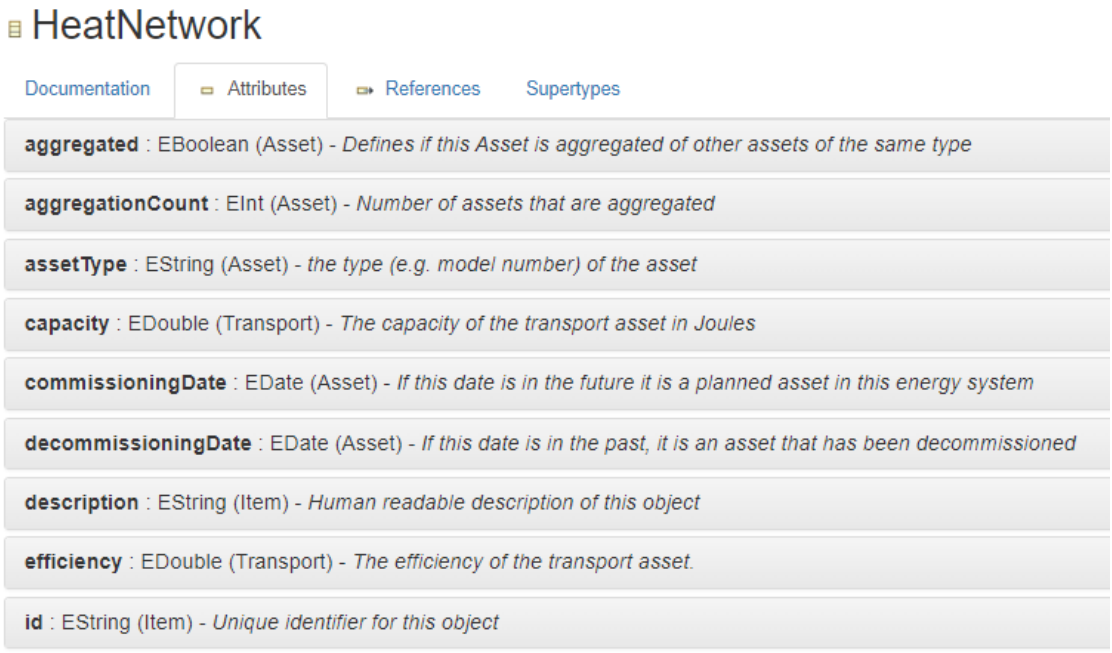

Figure 7: Example documentation of an energy asset in the *ESDL Model Reference Documentation*

3.2 Integration ESDL in AnyLogic

Figure 8 provides an overview of how the overall functional architecture looks like. It also shows what the main features of the different components are. Ideally, a bi-directional information exchange between AnyLogic and ESDL is desired. However, in this project, only the importer has been realised. An exporter could process output data of AnyLogic models in (future) ESDL supported programs, thus increasing the significance and value of the total energy model.

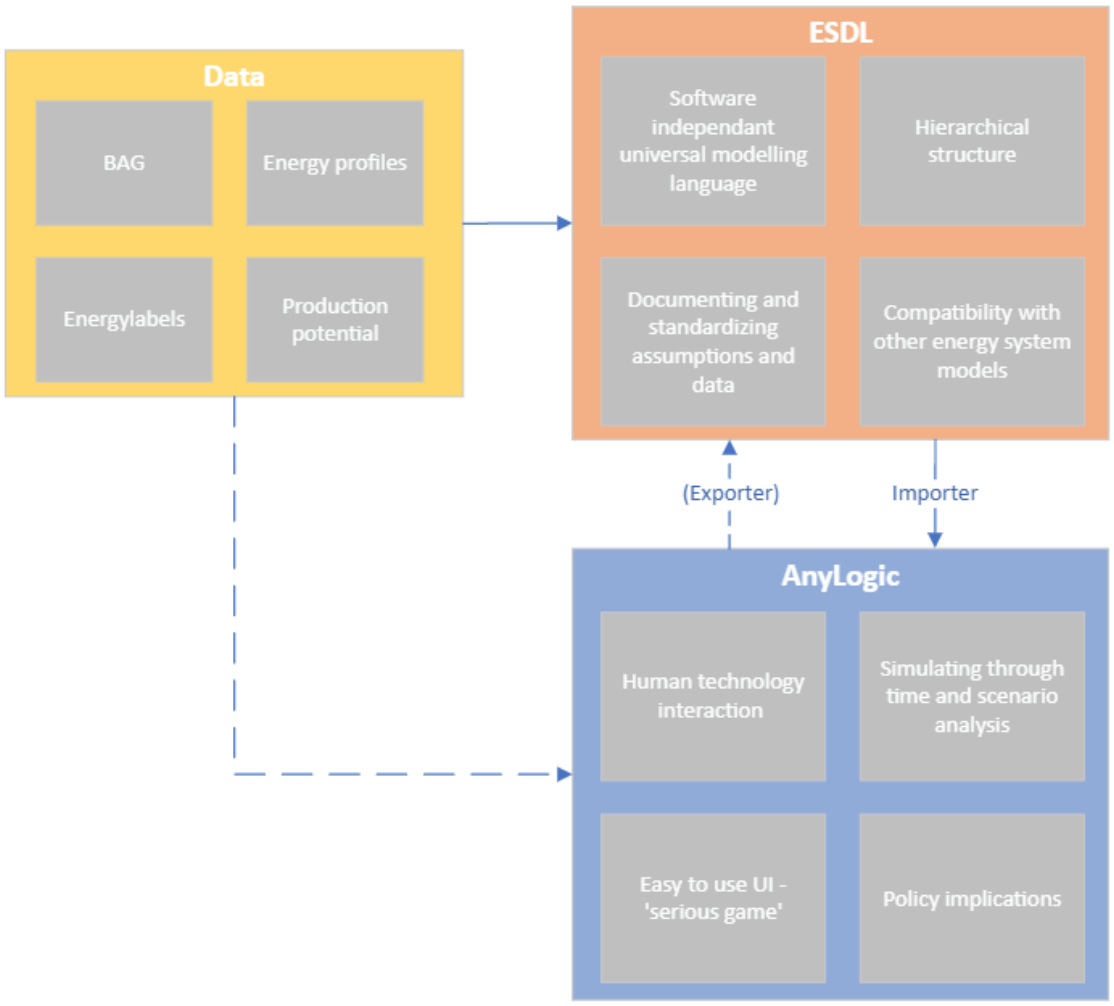

Figure 8: Overview synthesis of ESDL and AnyLogic

In appendix A the Java-based importer of ESDL in AnyLogic can be found. Comments are added in the code as an explanation of the functioning of it. On startup of an AnyLogic model, this script should be initiated. It opens and reads the ESDL files, but it does not store them as agents yet. With the creation of functions in the AnyLogic Main agent the ESDL data can be stored in energy asset specific agents. With the ESDL data in these agents, one can draw them on the geographical map and make much more complex energy systems by including features such as human technology interaction and user-friendly UI (as shown in the blue box in figure 8). In figure 9 a step by step overview is given of how the loading and processing of ESDL data works in AnyLogic. This can be very helpful if one wants to add other energy assets to the importer.

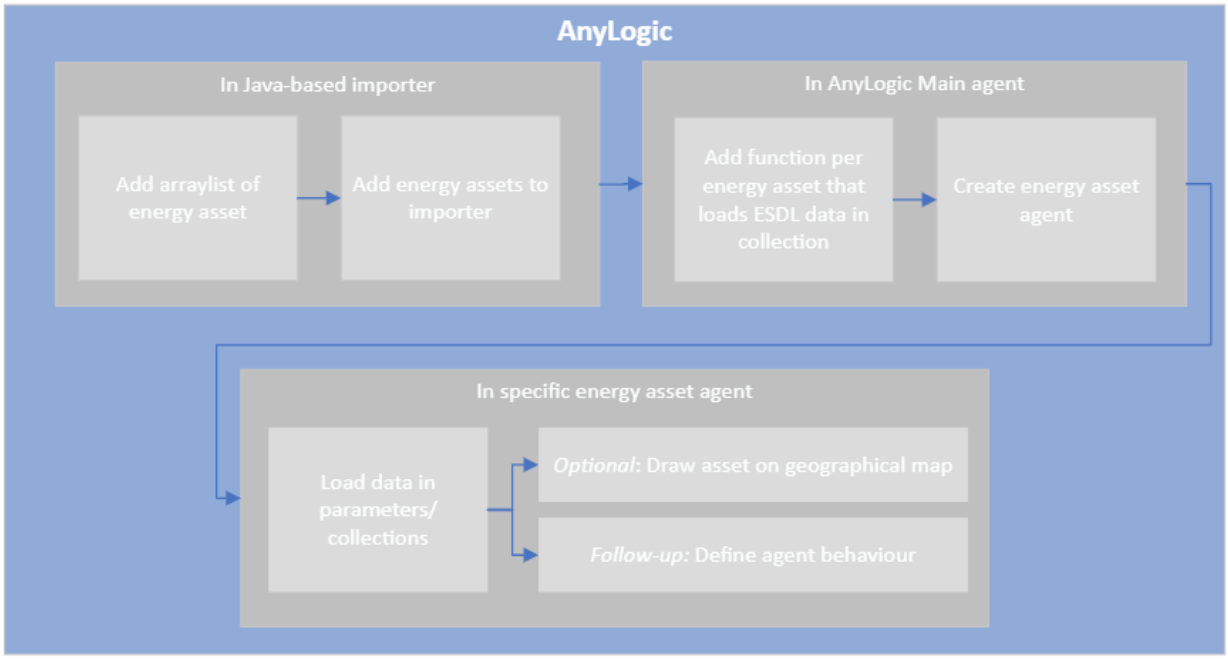

Figure 9: Step by step overview of ESDL data loading in AnyLogic

For being able to use the ESDL services in AnyLogic, a number of Java packages need to be loaded in AnyLogic. These dependencies are stored in so-called JAR files [TNO, 2022h]. Loading these packages allow for using the hierarchical structure of ESDL in AnyLogic.

The main features of the importer are listed below.

- Only the desired data will be loaded in from ESDL in AnyLogic, therefore complex ESDL files with many more properties than required are still handled properly in the importer script.
- If one wishes to import extra energy assets in AnyLogic, they should be added in the Java importer script. Besides this, a simple AnyLogic function needs to be created similarly to the existing ones and an agent in which the data will be saved will have to be initiated. These are relatively easy steps that don't take much time.
- The combination of both previous properties mean that by making just once a very extensive importer that handles virtually all ESDL assets, one can import the most simple to complex energy systems. This would make it very generic for all future simulations projects.
- Scaling up or down (for instance the simulation of an energy system of a single house to that of a whole province) is something that the importer handles appropriately, as ESDL prescribes these energy systems through the same hierarchical structure.
- The design of an energy system in a whole different area can easily be done by setting two calibration points to the proper coordinates. This makes the importer versatile and generic for a wide range of cases.

3.2.1 Application to the Apeldoorn-Noord case

The importer has been applied to the Apeldoorn-Noord case for testing purposes, as explained in the methodology (section 2). The industrial area designed in an ESDL supported program (the ESDL MapEditor) of figure 5 has been run through the AnyLogic importer. For this specific case, the assets and their relevant properties (as shown in table 1) are imported. This results in the imported ESDL assets as shown in figure 10. This figure displays the imported assets on a geographical map in AnyLogic.

Imported from ESDL	**Relevant properties**
Buildings	BAG data: - Building year - Floor area - BAG building ID - Building coordinates
Building unit	Building type
Heat network	Coordinates
Heatpump	Coordinates Power
Heat storage	Coordinates Capacity
Heating demand	Coordinates Power

Table 1: Assets, including their properties, that are implemented in the current importer

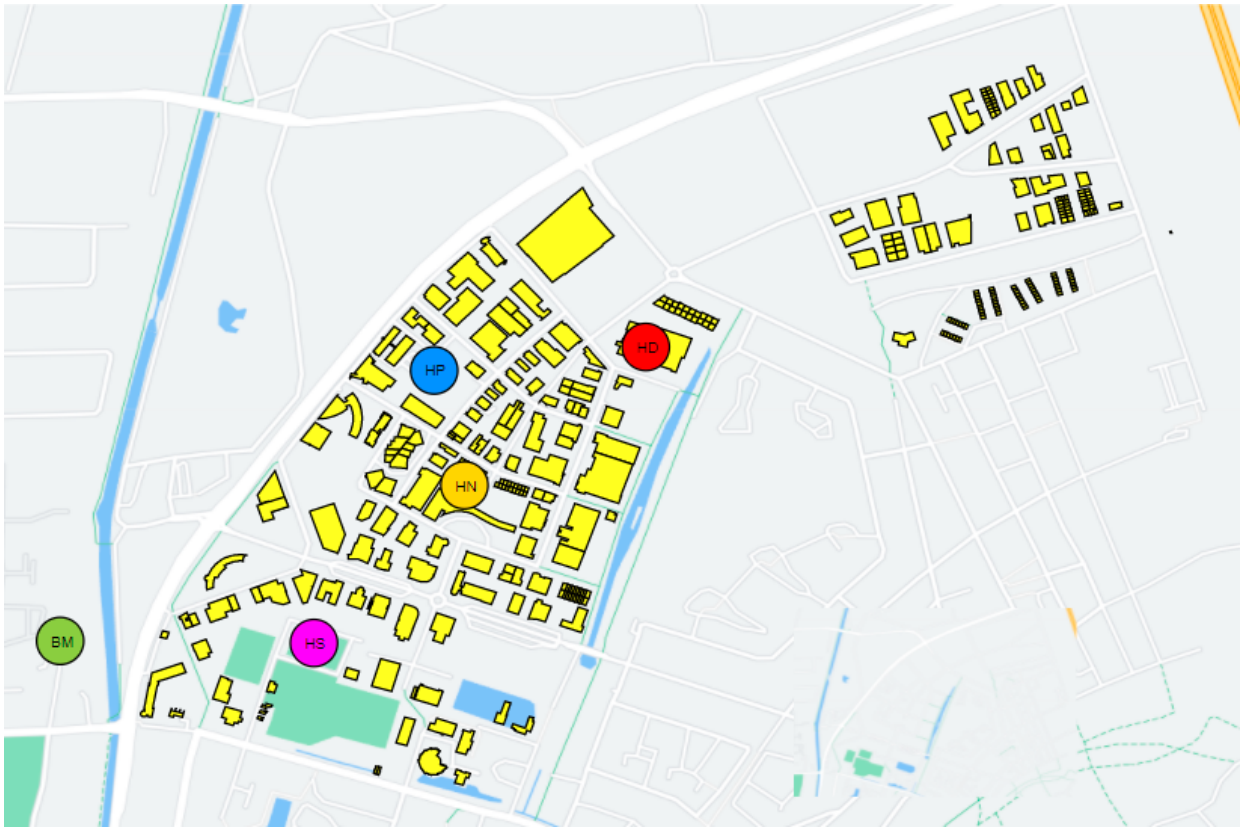

Figure 10: Energy system in Apeldoorn-Noord (see figure 5) imported in AnyLogic

Figure 11 shows an example of an agent (in this case the building agent) with its properties that are imported from the ESDL file. The functions that are needed for creating the imported file are shown on the left, the collections that help sort the data are shown in the middle while the individual imported parameters are shown on the right. In this case, the building has a floor area of 2211 m^2, is built in the year 2003 and has the 'shopping' building type.

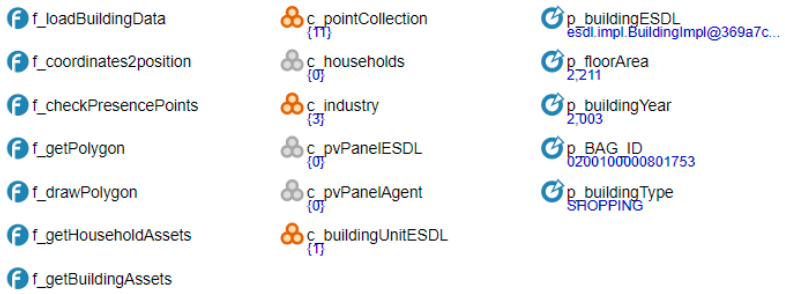

Figure 11: The building agent including the imported ESDL properties in AnyLogic from the Apeldoorn-Noord case

4 Conclusions and recommendations

In this research the possibilities and opportunities of integrating Energy System Description Language characteristics and data files in energy systems designed in AnyLogic are explored. By studying the workings of both ESDL and AnyLogic, the potential of this integration is mapped. Furthermore, the designed Java-based importer for ESDL in AnyLogic lays the foundation for a broad and universal importer that can handle any energy system efficiently and robustly.

The integration of ESDL provides a number of advantages and opportunities for ZEnMo Simulations. The standardization for energy system models leads to compatibility with and comparability to other energy system models. The ESDL importer for AnyLogic designed in this project proves that integration is possible and already shows a number of benefits in the design process of an energy model such as scalability and adaptability to other, more complex, energy system modelling cases. Besides this, the possibility for policy makers to explore their own energy system cases without the necessity to fully understand complex agent based models in Java is a huge potential of the synthesis between ESDL and AnyLogic.

The ESDL importer designed in this research is primarily a data importer, with less focus on the other ESDL characteristics such as standardization and structurization of energy models. It should be noted that the same data can be imported in AnyLogic through other ways as well. However, even with just the data importer, these characteristics are incorporated to a certain extent. For instance, if more energy assets and especially their properties are imported, naturally the standardized assumptions and data of ESDL will be incorporated in AnyLogic as well due to the replication of the ESDL standards. The same reasoning applies to that for the structurization of energy system models in AnyLogic. This, in combination with the aforementioned compatibility, comparability, adaptability and scalability, is in the core what the true value is of the synthesis of ESDL and AnyLogic.

There are some limitations that should be noted. These are primarily limitations to the possibilities and functionalities of ESDL. ESDL is still in a pretty early phase of development; bugs, imperfections and inconsistencies in ESDL and especially its supporting programs are still quite common. For instance, for scaling up to larger areas some bottlenecks can be encountered as the MapEditor is not yet robust enough to load data for large cities. The developers at TNO are constant updating and tweaking properties and the functionalities of it. Some more time for the development of it might make it more robust. A missing functionality of ESDL that would be very convenient for ZEnMo Simulations would be the addition of data on energylabels. Besides this, lack of exhaustive documentation on ESDL and the necessity to request for an account in the MapEditor makes it hard for inexperienced and new energy modellers to dive into the subject of ESDL and adopt it in their modelling projects.

As ESDL becomes more mature, it would be recommended to implement ESDL importers step by step more in the design process of an energy system, while ESDL exporters can be implemented afterwards. The adoption of ESDL by more and more parties in the field of energy modelling in the Netherlands (and possibly in the future abroad as well) indicates a strong relevance for, and interest by, the industry. It would be wise for ZEnMo Simulations to follow this trend.

References

[TNO, 2022a] TNO (2022a). Design principles. https://energytransition.gitbook.io/esdl/esdl-concepts/design-principles.

[TNO, 2022b] TNO (2022b). Eclipse documentation. https://energytransition.gitbook.io/esdl/how-to-use-esdl/using-esdl-to-model-an-energy-system.

[TNO, 2022c] TNO (2022c). Energy data modelling. https://energytransition.gitbook.io/esdl/energy-data-modelling.

[TNO, 2022d] TNO (2022d). Esdl mapeditor documentation. https://esdl-mapeditor-documentation.readthedocs.io/en/latest/index.html.

[TNO, 2022e] TNO (2022e). Esdl model reference documentation. https://energytransition.github.io/index.html.

[TNO, 2022f] TNO (2022f). Essim documentation. https://essim-documentation.readthedocs.io/en/latest/index.html.

[TNO, 2022g] TNO (2022g). Grip op de energietransitie met esdl. https://www.tno.nl/nl/aandachtsgebieden/informatie-communicatie-technologie/expertisegroepen/monitoring-control-services/grip-op-de-energietransitie-met-esdl/.

[TNO, 2022h] TNO (2022h). Integration with java. https://energytransition.gitbook.io/esdl/software-development/code-generation-java#integration-with-java.

[TNO, 2022i] TNO (2022i). Wat is esdl? https://www.esdl.nl/geinteresseerden.html.

[TopSectorEnergie, 2022] TopSectorEnergie (2022). Models and data interfaces for energy – proof of concept. https://projecten.topsectorenergie.nl/projecten/models-and-data-interfaces-for-energy-proof-of-concept-32990.

Appendix A - Java importer

```java
// Import packages
import java.io.IOException;
import java.util.HashMap;

import org.eclipse.emf.common.util.EList;
import org.eclipse.emf.common.util.URI;
import org.eclipse.emf.ecore.EObject;
import org.eclipse.emf.ecore.xmi.XMIResource;
import org.eclipse.emf.ecore.xmi.impl.XMIResourceImpl;

import esdl.Area;
import esdl.Asset;
import esdl.BiomassHeater;
import esdl.Building;
import esdl.BuildingUnit;
import esdl.ElectricityNetwork;
import esdl.EnergySystem;
import esdl.EsdlPackage;
import esdl.HeatNetwork;
import esdl.HeatPump;
import esdl.HeatStorage;
import esdl.HeatingDemand;
import esdl.Instance;
import esdl.PVPanel;

public class ESDLImporter {

        // Create list of imported assets from AnyLogic
        ArrayList<Building> c_building = new ArrayList<Building>();
        ArrayList<BuildingUnit> c_buildingUnits = new ArrayList<BuildingUnit>();
        ArrayList<ElectricityNetwork> c_electricityNetworks = new ArrayList<
            ElectricityNetwork>();
        ArrayList<HeatNetwork> c_heatNetworks = new ArrayList<HeatNetwork>();
        ArrayList<HeatStorage> c_heatStorages = new ArrayList<HeatStorage>();
        ArrayList<HeatingDemand> c_heatingDemands = new ArrayList<HeatingDemand>();
        ArrayList<HeatPump> c_heatPumps = new ArrayList<HeatPump>();
        ArrayList<PVPanel> c_pvPanels = new ArrayList<PVPanel>();
        ArrayList<BiomassHeater> c_biomassPlant = new ArrayList<BiomassHeater>();


        /***
         * Constructor: Aim of this script is to print a message about every asset in
         * the ESDL file.
         *
         * @param esdlFileName
         * @throws IOException
         */

        public ESDLImporter(String esdlFileName) throws IOException {

                System.out.println("Loading ESDL from file: " + esdlFileName);

                // Load ESDL File into the EnergySystem object
                EnergySystem energySystem = (EnergySystem) loadEcoreResource(esdlFileName);

                // @formatter:off
                //
                // ESDL Structure:
                // ---------------
                // Energy System
                // |- Energy System Instance
                //                 |- Area
                //                         |- Sub-area/Building (Optional)
                //                                 |- Assets (PV Panels, Demands, etc.)
                //                         |- Assets (PV Panels, Demands, etc.)
                // Start from outermost elements and work your way in
```

```
66                  //
67                  // @formatter:on
68
69                  // Grab all instances of the EnergySystem
70                  EList<Instance> instances = energySystem.getInstance();
71                  for (Instance instance : instances) {
72                          if (instance != null) {
73                                  //System.out.println("Processing instance with id: " +
                                        instance.getId());
74                                  // Each instance has a root area
75                                  Area area = instance.getArea();
76                                  if (area != null) {
77                                          // Process this area recursively (and print the
                                                asset indented accordingly)
78                                          processAssetsInArea(1, area);
79                                  }
80                          }
81                  }
82          }
83
84          /**
85           * Process assets in an area. Remember that an area can contain sub-areas
86           *
87           * @param indent
88           *            How many spaces to indent printing of the assets in this area by
89           * @param area
90           *            Area object to process
91           */
92          public void processAssetsInArea(int indent, Area area) {
93
94                  // For each sub-area (if any) in Area, process assets
95                  for (Area subArea : area.getArea()) {
96                          processAssetsInArea(indent + 1, subArea);
97                  }
98
99                  // Process assets contained in this area
100                 processAssets(indent + 1, area.getAsset());
101         }
102
103         /***
104          * Process assets from a list of assets. EList is an ECore wrapper around a Java
105          * list, so you can use normal list functions.
106          *
107          * @param indent
108          *            How many spaces to indent printing of the assets in this area by
109          * @param assets
110          *            List of assets
111          */
112         public void processAssets(int indent, EList<Asset> assets) {
113
114                 // For all assets that are imported in list, do:
115                 for (Asset asset : assets) {
116
117                         // If the asset is a building, it may contain assets inside of it.
118                         // So recursively process the assets inside this building (if any)
                                first.
119                         if (asset instanceof Building) {
120                                 Building building = (Building) asset;
121                                 c_building.add(building);
122                         }
123                         // If asset is a building unit, the same rule as with buildings
                                apply.
124                         // Recursively process the assets inside this building unit (if any)
                                .
125                         else if (asset instanceof BuildingUnit) {
126                                 BuildingUnit buildingUnit = (BuildingUnit) asset;
127                                 c_buildingUnits.add(buildingUnit);
128
```

```
129                             processAssets(indent + 1, buildingUnit.getAsset());
130                     }
131
132                     // For all other assets, do:
133                     else if (asset instanceof ElectricityNetwork) {
134                             ElectricityNetwork electricityNetwork = (ElectricityNetwork)
                                    asset;
135                             c_electricityNetworks.add(electricityNetwork);
136                     }
137
138                     else if (asset instanceof HeatNetwork) {
139                             HeatNetwork heatNetwork = (HeatNetwork) asset;
140                             c_heatNetworks.add(heatNetwork);
141                     }
142
143                     else if (asset instanceof HeatStorage) {
144                             HeatStorage heatStorage = (HeatStorage) asset;
145                             c_heatStorages.add(heatStorage);
146                     }
147
148                     else if (asset instanceof HeatingDemand) {
149                             HeatingDemand heatingDemand = (HeatingDemand) asset;
150                             c_heatingDemands.add(heatingDemand);
151                     }
152
153                     else if (asset instanceof HeatPump) {
154                             HeatPump heatPump = (HeatPump) asset;
155                             c_heatPumps.add(heatPump);
156                     }
157
158                     else if (asset instanceof PVPanel) {
159                             PVPanel pvPanel = (PVPanel) asset;
160                             c_pvPanels.add(pvPanel);
161                     }
162
163                     else if (asset instanceof BiomassHeater) {
164                             BiomassHeater biomassPlant = (BiomassHeater) asset;
165                             c_biomassPlant.add(biomassPlant);
166                     }
167
168                     else {
169                             String assetClass = asset.getClass().getInterfaces()[0].
                                    getSimpleName();
170                             // Remaning assets that are not imported are printed (this
                                    code can also be switched off):
171                             String message = "Found Asset of type " + assetClass + "
                                    with id: " + asset.getId();
172                             System.out.print(message);
173                     }
174             }
175     }
176
177     /***
178      *
179      * Function to load ESDL file into an EnergySystem object.
180      *
181      * @param <T>
182      *             Type of ECore object at the root of the XML (In our case, this is
183      *             EnergySystem)
184      * @param filepath
185      *             Path to ESDL File
186      * @return ECore object of type T (i.e, EnergySystem)
187      * @throws IOException
188      */
189     @SuppressWarnings("unchecked")
190     public static <T extends EObject> T loadEcoreResource(String filepath) throws
            IOException {
191             EsdlPackage.eINSTANCE.eClass();
```

```
192                 URI fileURI = URI.createFileURI(filepath);
193                 XMIResourceImpl resource = new XMIResourceImpl(fileURI);
194                 resource.getDefaultLoadOptions().put(XMIResource.
                        OPTION_DEFER_IDREF_RESOLUTION, Boolean.TRUE);
195                 resource.setIntrinsicIDToEObjectMap(new HashMap<String, EObject>());
196                 resource.load(null);
197                 return (T) resource.getContents().get(0);
198         }
199
200         /**
201          * Main method
202          *
203          * @param args
204          * @throws IOException
205          */
206         public static void main(String[] args) throws IOException {
207         }
208
209 }
```