EMBRY-RIDDLE Aeronautical University™
SCHOLARLY COMMONS

International Journal of Aviation, Aeronautics, and Aerospace

Volume 5 | Issue 5

Article 4

2018

SES and Ecore for Ontology-based Scenario Modeling in Aviation Scenario Definition Language (ASDL)

Shafagh Jafer
Embry-Riddle Aeronautical University, jafers@erau.edu
Bharvi Chhaya
Embry-Riddle Aeronautical University, chhayab@my.erau.edu
Bernard P. Zeigler
RTSync Corp, zeigler@rtsync.com
Umut Durak
German Aerospace Center, umut.durak@dlr.de

Scholarly Commons Citation

Jafer, S., Chhaya, B., Zeigler, B. P., & Durak, U. (2018). SES and Ecore for Ontology-based Scenario Modeling in Aviation Scenario Definition Language (ASDL). *International Journal of Aviation, Aeronautics, and Aerospace, 5*(5). Retrieved from https://commons.erau.edu/ijaaa/vol5/iss5/4

Introduction

Aviation has long been benefiting from modeling and simulation for technology development, testing, training and integration purposes. Although the importance of scenarios in this domain has been well known, there still exists a lack of common understanding and standardized practices in aviation simulation scenario development (Durak, Topçu, Siegfried, & Oguztuzun, 2014; Jafer, Chhaya, Durak, & Gerlach, 2016). It is an extensive process beginning with the stakeholders' descriptions of the scenario and finishing with the generation of the corresponding executable specifications (Durak et al., 2014). Simulation scenario can be defined as the specification of initial and terminal conditions, significant events and the environment, as well as the major entities, their capabilities, behavior and interactions over time (Department of Defense, 1998). With one sky shared globally, the next generation of aviation technologies call for immediate action on defining a standardized mechanism for developing, sharing, and integrating large-scale simulation scenarios among global stakeholders. With the help of model- and simulation-based engineering, large-scale systems integration and demonstrations take place seamlessly. This demands for common understanding of simulation scenarios, allowing for cross-platform interoperability such that scenarios can be run on any simulator worldwide.

Developing a scenario definition language for a specific domain has been recently conducted for military simulations. Military Scenario Definition Language (MSDL) (Wittman, 2009) was developed and published as a standard by Simulation Interoperability Standards Organization (SISO; 2008). Similarly, the recent effort published at American Institute of Aeronautics and Astronautics (AIAA), as reported by Jafer, Chhaya, and Durak (2017a), proposes to standardize aviation scenario development through Aviation Scenario Definition Language (ASDL) to allow the global aviation Modeling and Simulation (M&S) community, from academia, industry, and government agencies, benefit from a common scenario definition platform, enabling model transformation, reusability, and interoperability across various simulation environments. ASDL was proposed with the following goals in mind:

1. A common mechanism (published standards) for specifying, verifying and executing aviation scenarios.

2. The ability to create platform-independent aviation scenario that can be shared between simulation environments and various simulators.

3. A way to improve scenario consistency among globally collaborative simulations.

4. A platform for coupling training needs with an efficient scenario generation process.

5. The ability to reuse aviation scenarios as scenario descriptions in many areas within aviation, e.g., technology and product development and integration (US and European aviation programs) or flight training, etc.

6. Reusability and adaptation to other transportation areas such as ground, water, and even space exploration.

ASDL takes scenario-based development as the core activity in constructing a formal scenario definition language. Concepts such as ontology-based and model-based development have been highly utilized to incorporate automated transformations and executable generation (Saeki & Kaiya, 2006). At the core of every domain-specific language exists an ontology that captures all domain's key terminology and relationships. The elements of model-driven methodology are modeling languages, metamodels, and transformations (Brambilla, Cabot, & Wimmer, 2012). Modeling languages enable the definition of a concrete representation for a model and metamodels are used to define modeling languages. Transformations are described as the mappings between models which are specified at metamodel level.

Constructing an ontology has been addressed in the literature through a number of techniques (Chandrasekaran, Josephson, & Benjamins, 1999; Farquhar, Fikes, & Rice, 1997; Maedche & Staab, 2001). To develop ASDL's ontology, two approached were utilized: (1) metamodeling with Eclipse Modeling Framework (EMF) Ecore, and (2) metamodeling with System Entity Structure (SES).

EMF is a framework within the Eclipse ecosystem for Model-Driven Development (MDD) (Steinberg, Budinsky, Merks, & Paternostro, 2008). EMF core (Ecore) is a standard for data models that offers a metamodel for describing models as well as a persistence support with the ability to export results to eXtensible Markup Language (XML) format. On the other side, SES is a high-level ontology which was introduced to specify a set of system structures and parameter settings. It has long been used for modeling variable structure systems and recently applied to problems of model-based simulation system engineering such as model-based testing (Durak, Schmidt, & Pawletta, 2015; Schmidt, Durak, & Pawletta, 2016) and variability management (Pawletta, Schmidt, Zeigler, & Durak, 2016).

This paper proposes a model-based simulation scenario development approach using SES. In order to do this, first, simulation scenario development will be introduced. The target language ASDL will then be described in detail including the metamodel and its implementation. Following this discussion, the proposed approach using SES will be presented and discussed. As an application example, we then provide a case where our proposed SES/Ecore approach is utilized in building a scenario-driven training toolset for air traffic controllers at the Federal Aviation Administration (FAA) Academy.

Literature Review

Simulation Scenario Development

Simulation scenario can be defined as the specification of initial and terminal conditions, significant events and the environment as well as the major entities, their capabilities, behavior, and interactions over time (Department of Defense, 1998). Although the importance of scenarios in M&S has long been well known, there still exists a lack of common understanding and standardized practices in simulation scenario development. Based on the North Atlantic Treat Organization (NATO) Guideline on Scenario Development (NATO, 2015), three types of scenarios are produced in successive stages of the scenario development process. These scenarios are: operational scenarios, conceptual scenarios, and executable scenarios (Siegfried et al., 2012, 2013), illustrated in Figure 1.

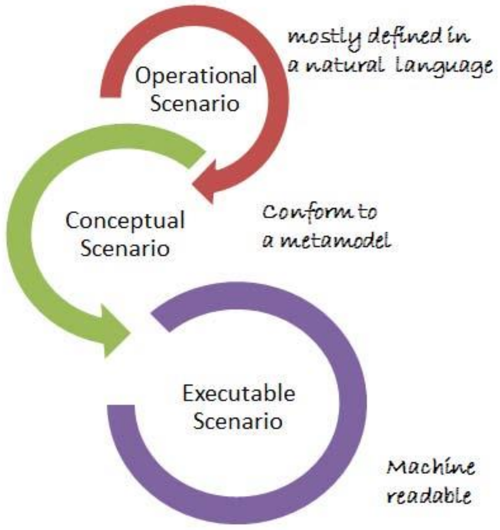


Figure 1. Three types of simulation scenarios. Adapted from “Scenarios in Military (distributed) Simulation Environments” by R. Siegfried et al., 2012, *Science and Technology Organization*, TO Technical Report TR-MSG-086-Part-II.

Operational scenarios are described in the early stages by the domain expert in the form of natural language, either oral or written. The key elements an operational scenario are the entities, their initial states and the events. The key element in a flight simulation scenario is the Aircraft. An example aircraft landing operational scenario is given as follows:

A normal landing scenario starts with aircraft ER-1234 approaching at Daytona Beach Airport (DAB) originating from Atlanta Airport (ATL). Stable weather conditions nearing Daytona Beach are reported as cross wind: 77, dew point: 60, sky condition: few clouds at 5500 feet, temperature: -44, visibility: 10, wind shear: 11.8. While cruising, the pilot requests descent. The ATC controller at DAB, grants the request and notifies the pilot to land on runway 7L. The pilot initiates descend and reports Aircraft status of altitude: 34000ft, latitude: 82.35, longitude: 31.49, flight rules: VFR, and ground speed: 543. The aircraft lands on Runway 7L.

The operational scenarios provide a coarse description of the intended situation and its dynamics, but they need to be refined and augmented with additional information pertaining to simulation. This refinement is usually done by the simulator experts and results in conceptual scenarios. Conceptual scenarios specify the piece of the world to be represented in the simulation environment in detail. They should incorporate all crucial information for executing the operational scenario. On the other hand, the executable scenario is the specification of the conceptual scenario in a particular format in order to be processed by the simulation applications for initialization, and execution. They support scenario management activities such as scenario distribution and role casting (Topçu, Durak, O˘guztüzün, & Yilmaz, 2016). For this purpose, the conceptual scenarios need to be transformed into executable scenarios. The transformation from conceptual scenarios to executable scenarios is undertaken primarily by simulator experts. Ideally, the resulting executable scenarios are specified in a way that they can directly be processed by the target simulator.

Domain-Specific Language

Domain-Specific Language (DSL) is a custom-tailored computer language for a particular application domain (Fowler, 2010). DSL is created to specifically target problems in a specific domain, and stresses upon the main ideas, features, constraints, and characteristics of that domain. DSL enables developers to construct models that are specific to their application. These models are mainly composed of elements and relationships that are verified to be valid for that application.

DSLs allow users to write complete application programs for the given domain more quickly and more effectively than they can with a general-purpose language (GPL). A well-designed DSL intends to capture precisely the semantics of an application domain (Hudak, 1997). Advantages of programs written in DSLs as compared to GPLs are that: they are more concise, they can be written more quickly, they are easier to maintain, and they can be written by non-programmers.

The greatest benefit of DSL is that it allows non-developers and those who are not experts in the domain to understand the overall design. This is normally supported by allowing graphical modeling, usually in the form of a drag and drop capability to construct models. DSL augmented with model-to-text transformation capabilities directly allows for automatic generation of source code from model.

Ontology

An ontology describes the concepts and relationships that are important in a particular domain, providing a vocabulary for that domain as well as a computerized specification of the meaning of terms used in the vocabulary (Gruber, 1993). One of the benefits of using ontologies is their capacity to be easily extended using new knowledge generated by experts so all existing ontologies can be used as a starting point for further development (Hilera & Fernández-Sanz, 2010). Among the existing ontology specification frameworks, the Web Ontology Language format (OWL) is most commonly used by the DSL community. OWL enables describing a domain in terms of classes, properties and individuals and may include rich descriptions of the characteristics of those objects (Bechhofer, 2009; McGuinness, 2004).

Ontology-Driven Software Development (ODSD) has emerged as a significant mechanism in creating domain-specific languages (Ceh, Crepinšek, Kosar, & Mernik, 2011), allowing for expressing domain concepts effectively (Pan, Staab, Aßmann, Ebert, & Zhao, 2012). Ontology provides a quick and simplified description of a DSL, abstracting language's technically details, while highlighting key terminology and specifics. Once an ontology is built, it is a simple process to generate the language's metamodel and establish relationships among related concepts. An automated process that takes in DSL's ontology and generates its corresponding metamodel sounds highly efficient. This has been studied in various development environments including EMF (Jafer, Chhaya, & Durak, 2017b).

Target Language: Aviation Scenario Definition Language (ASDL)

A well-defined language for aviation scenario specification would enable the reuse of scenarios among different simulators. To address aviation simulations limitations, Jafer et al. (2016) introduced the Aviation Scenario Definition Language (ASDL) which aims to provide a standard aviation scenario specification mechanism. Based on DSL design methodologies, ASDL provides a well-structured definition language to define aviation mission scenarios. ASDL supports verifying and executing aviation scenarios, effective sharing of scenarios among various simulation environments, improving the consistency among different simulators and enabling the reuse of scenario specifications. By taking a formal approach in defining aviation scenarios, ASDL provides consistency and completeness checking, and model-to-text transformations capabilities for various targets in the aviation domain. Built in EMF (Steinberg et al., 2008), ASDL tool suite can also support a graphical modeling environment to automatically transform scenario models into executable scenario scripts (Jafer et al., 2017a). The current version of ASDL supports specification of departure, re-route, and landing scenarios (“ASDL Ontology,” 2016).

ASDL Ontology

To capture aircraft landing details, it is essential to have a definitions reference list that highlights all key terminology as well as procedures and operations that are communicated between the pilot and ATC. The United States’ FAA and the Single European Sky ATM Research (SESAR) programs provide inclusive glossaries that provide key terminology and concept of operations (FAA Flight Standards Service AFS Flight Program Division, 2012; SESAR, 2015). A review of existing ontologies resulted in one aviation ontology being discovered. However, this described the structural and physical entities of an aircraft, and hence did not provide any useful terms that could be reused. Thus, a new aviation-specific ontology was created for this project.

ASDL ontology consists of two parts: keywords that describe the physical model and operation of flights, and words that describe key communication between the control tower and pilots. This section lists majority of these keywords along with their definitions and use. A complete ASDL ontology can be accessed online (“ASDL Ontology,” 2016). Once sufficient keywords were identified, the primarily used terms were added to a basic ontology created using Protégé (Alatrish, 2013), which saves them in OWL format. Protégé is an ontology development environment that makes it easy to create, upload, modify, and share ontologies for collaborative viewing and editing (Musen, 2015). The Web Ontology Language (Bechhofer, 2009) is a language for defining ontologies on the Web. An OWL ontology describes a domain in terms of classes, properties and individuals and may include rich descriptions of the characteristics of those objects (Stanford Center for Biomedical Informatics Research, 2017).

An ontology focuses mainly on classes which describe the concepts of the domain. It follows a hierarchical model where subclasses are all necessarily a part of the superclass (Noy & McGuinness, 2001). The ASDL ontology has four base classes: Air_Traffic_Control, Aircraft, Airport, and Weather. This can be seen in Figure 2, created by the authors. All these terms have been defined in Table 1.

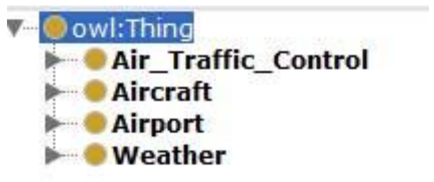


Figure 2. High-level view of ASDL ontology.

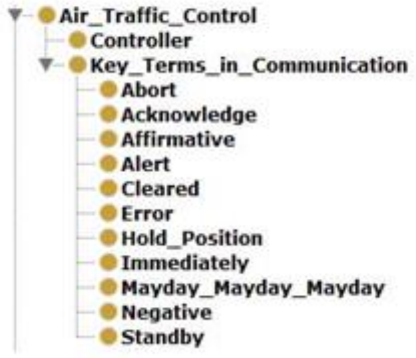
Table 1

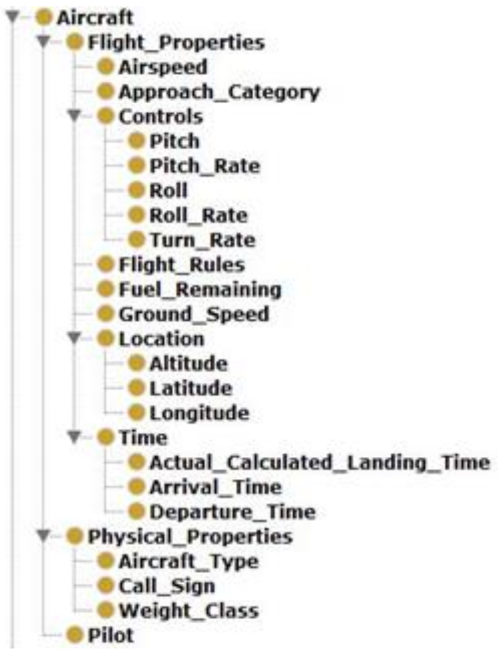
Definition of terms in base class of ASDL Ontology.

Term	Definition
Air Traffic Control	A service operated by appropriate authority to promote the safe, orderly and expeditious flow of air traffic.
Aircraft	Any machine that can derive support in the atmosphere from the reactions of the air other than the reactions of the air against the earth’s surface.
Airport	An area on land or water that is used or intended to be used for the landing and takeoff of aircraft and includes its buildings and facilities, if any.
Weather	The state of the atmosphere at a place and time as regards heat, dryness, sunshine, wind, rain, etc.

Note: Adapted from “ASDL Ontology” by GitHub, 2016, p. 1.

As shown in Figure 3a, the ATC class includes a Controller and various key terms that need to be used in conversation. The main part of this ontology involves the aircraft and its properties. Figure 3b, created by the authors, shows the subclasses of the Aircraft class.

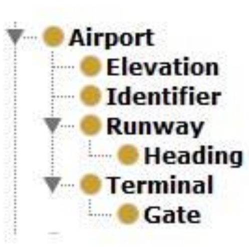


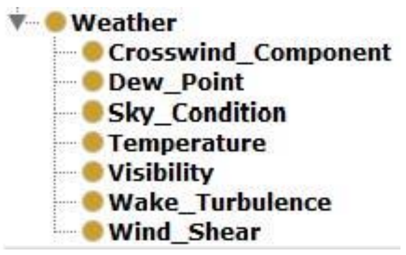


(a) Elements present in ATC class. (b) Elements present in Aircraft class.

Figure 3. Elements present in ATC and Aircraft classes in ontology.

The Flight_Properties subclass describes the rules (IFR or VFR) that govern the flight, the speed of the aircraft, the fuel remaining and has three other subclasses: controls (pitch, roll and turn rates), location (altitude, latitude, longitude) and time (arrival time, departure time, and ACLT). The physical properties subclass contains the call sign, type of aircraft and its weight class. The Airport class includes an identifier, the details of terminals and gates present in the airport, its elevation as well as runway details. Each runway's information also includes its heading. These author-created Figures can be seen in Figure 4a in a similar approach, 4b shows the items present in the Weather class.





(a) Elements present in Airport class. (b) Elements present in Weather class.

Figure 4. Elements present in ATC and Aircraft classes in ontology.

Protégé instances can also be created of all these classes having the requisite properties. There is a large scope for addition of various related items to the ontology; this is only a basic framework that lists the main items that are used in this model. A more comprehensive list of definitions is available online at ASDL Ontology (2016).

ASDL Metamodel in Ecore

Following the principles of MDD, scenario development takes place as the transformation of operational scenarios (defined in a natural language) to conceptual scenarios (conforming to ASDL formal metamodel) then to executable scenarios (specified using ASDL scenario definition). To capture all the necessary constructs for a simulation scenario, Simulation Interoperability Standards Organization (SISO) Base Object Model (BOM) (SISO Base Object Model Product Development Group, 2006) was adopted as the baseline metamodel. BOM is a standard that introduces the interplay, the sequence of events between simulation elements, as well as the reusable pattern, and provides a standard to capture the interactions. In ASDL, this baseline was extended to capture all the domain related concepts and terminology as constructs. Eclipse Modeling Framework (EMF) was used to create ASDL metamodel. EMF is a commonly used modeling framework and code generation facility for building tools based on metamodels (Gronback, 2014). Once a model specification has been described, EMF provides tools to produce a set of Java classes for the model, along with a set of adapter classes which enable viewing and editing of the model. The first step is to have a design of the structure of the data which includes all data items and the relationships between them. This can then be defined in EMF in the Ecore format, which is basically a subset of Unified Modeling Language (UML) Class diagrams. This is the metamodel, which describes the structure of the model. A metamodel can further be used to generate a model, which is a concrete instance of this structured data. The Ecore file allows users to define the following elements for the model:

1. EClass: a class with zero or more attributes and references.

2. EAttribute: an attribute of the class which has a name and a type.

3. EReference: an association between two classes.

4. EDataType: the data type of an attribute.

In ASDL, first an aviation scenario metamodel was developed in order to capture the necessary characteristics of a flight. This drew upon the ontology developed in the first part of the project in order to define these attributes. Second, the aviation metamodel was integrated with the BOM metamodel in order to define scenarios with specific aviation-related properties. Constructing ASDL metamodel was adapted from the framework introduced by Durak et al. (2014):

1. Define the classes required to accurately represent the model.

2. Determine the attributes used to describe the classes.

3. Define the structure and relationships between the classes.

4. Create an Ecore model based on the entities identified.

5. Integrate this model of aviation entities into the BOM framework.

6. Generate Java code for the model.

7. Create a runtime instance and use it to define and edit an aviation scenario.

The current ASDL model object allows users to define four different kinds of scenarios: departure, reroute, and landing. It also includes pilots, airports, runways, control towers, flight properties, weather patterns and aircrafts. This metamodel was integrated with the BOM entities of interplays, state machines and events in order to describe a flight scenario. This is seen in Figure 5.

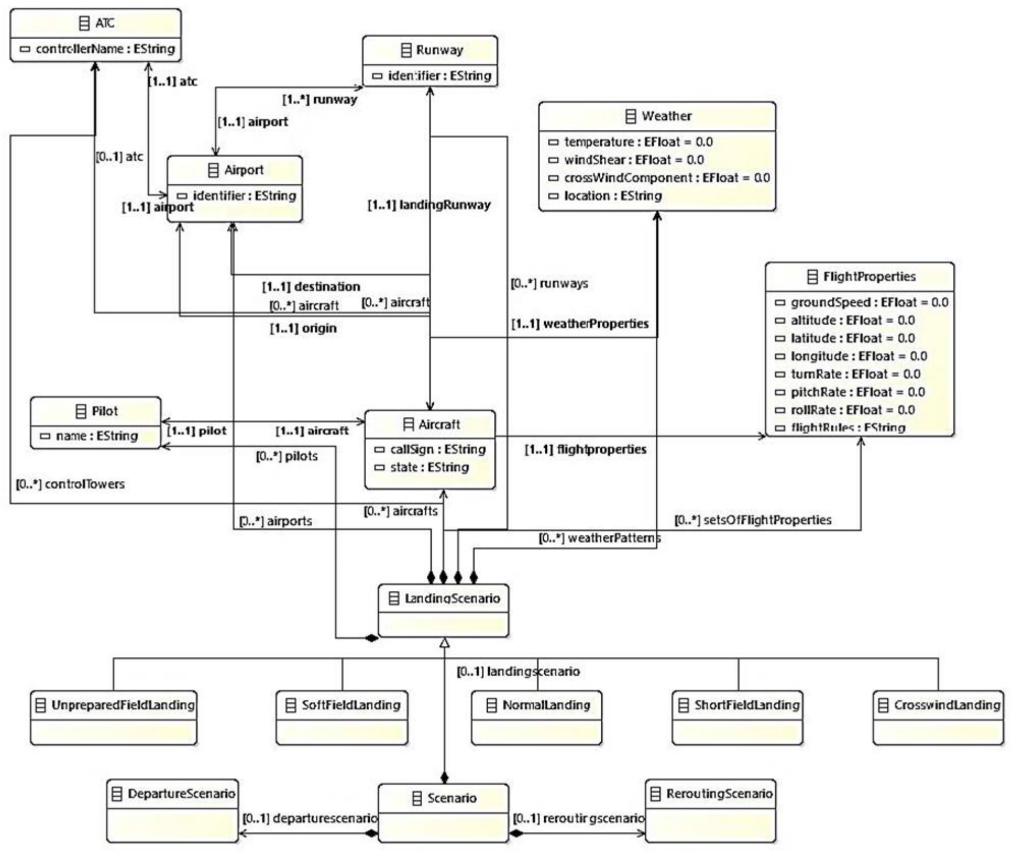


Figure 5. ASDL metamodel defined in EMF Adapted from "Formal Scenario Definition Language for Aviation: Aircraft Landing Case Study" by S. Jafer et al., 2016, *AIAA Modeling and Simulation Technologies conference*.

Approach: Ontology-based Scenario Development

To generate an executable simulation scenario from a given DSL ontology, a number of transformations must occur. Model transformations are an essential model-based development practice. These transformations allow for reflection of the data captured in one model to another as well as the addition of specialized information to the source model. Figure 6, created by the author, provides an overall illustration of our proposed approach, where an ASDL Suite provides an environment to specify a scenario from the early stage of scenario properties capturing, to providing an executable simulator-independent script.

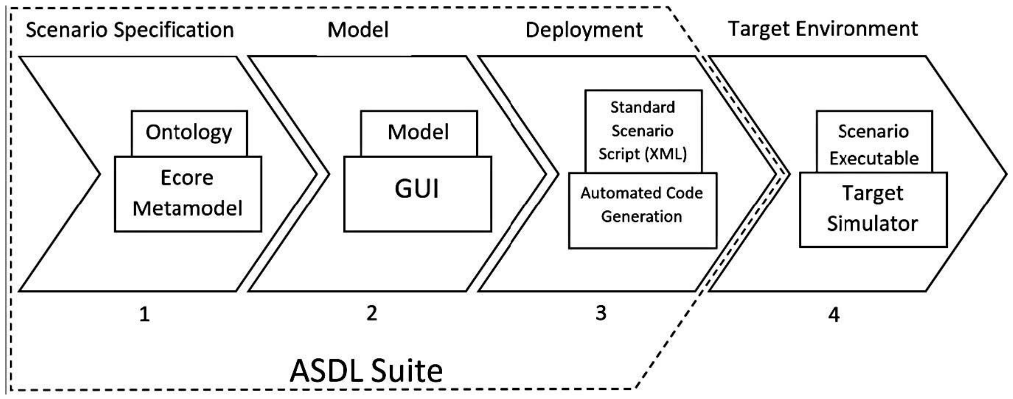


Figure 6. Ontology-driven scenario development with ASDL Suite.

The ASDL Suite comprises efforts in three stages. The first step is the specification of conceptual scenarios in the form of an Ecore metamodel created in Eclipse. This metamodel is built on top of an ontology, which captures all the keywords and concepts that define a flight scenario. In the next step, this metamodel is used to create a model of a specific scenario. This can be performed with the use of the ASDL Graphical User Interface (GUI), which allows a user to pick their required scenario elements from a menu. This facilitates the encapsulation of all metamodel and ontology details and allows the user to only interact with the parts of the tool they directly need, without being burdened with any background code. The deployment stage uses the information entered by the user into the model, and with automated code generation facilities, produces a standard scenario script in eXtensible Markup Language (XML) format. This XML script is then turned over to the end users for executing the scenario in their target simulator.

The following sections discuss the details of various transformations occurring in EMF environment, representing the required steps in taking an operational scenario and turning it into an executable simulator-specific script.

Automated Ontology to Metamodel Transformation

The automation of the mapping process from ontology to metamodel can be accomplished by creating an Eclipse plug-in that can read the Ontology files in OWL/XML format and convert them into Ecore objects using a set of established rules as has been shown in Figure 7. Fully automated transformation process is explained extensively in a previous work (Jafer et al., 2017b) where we discuss challenges and shortfall of such automation.

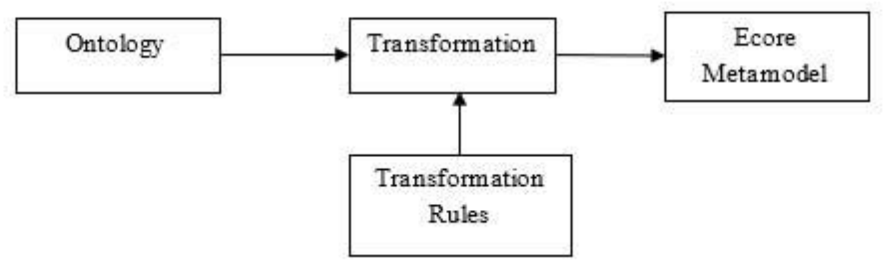


Figure 7. Transformation from ontology to metamodel. Adapted from "Owl Ontology to Ecore Metamodel Transformation for Designing a Domain Specific Language to Develop Aviation Scenarios" by S. Jafer et al., 2017b, *Proceedings of the Symposium on Model-Driven Approaches for Simulation Engineering*.

Text-to-Model Transformation

Text-to-Model (T2M) transformations are generally implemented for the purposes of reverse-engineering models from code and for creating models of existing legacy code (Bruneliere, Cabot, Jouault, & Madiot, 2010). T2M transformations are performed by using code to obtain the UML diagrams of a system. This is achieved with the use of a parser for the code along with some mechanism to extract the relationships present between elements of the code. It is a highly-complicated and challenging process to write a T2M transformation code for complex languages. However, Eclipse enables the transformation of any XML schema (which is an XSD file) into an Ecore metamodel (Budinsky, Steinberg, Ellersick, Grose, & Merks, 2004). In the case of ASDL, no T2M transformation was required as all modeling was directly performed using EMF.

Model-to-Text Transformation

Model-to-Text (M2T) transformations are mainly used to bridge the gap between the modeling language and the programming language by defining methods of automated code generation. Eclipse allows for the use of multiple tools in order to generate textual artifacts from models. Three major M2T transformation tools available within Eclipse are Acceleo, Xpand, and the Java Emitter Template (JET) (Skrypuch, 2007). EMF uses Java Development Tools (JDT) to build the editor within its code generation facility. The JET component's framework is used for automated transformations in EMF. In this case, JET was used to convert the ASDL metamodel into Java source code.

Model-to-Model Transformation

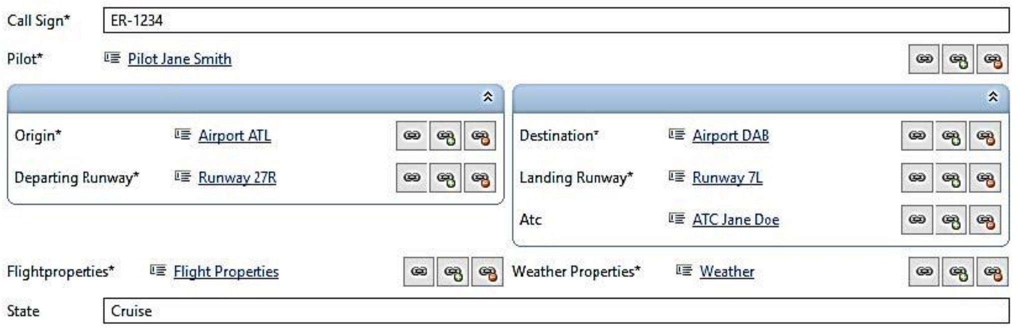
A Model-to-Model Transformation (MMT) has a model as both input and output, but with different parent metamodels. It accepts a model conforming to a particular metamodel and converts it into a model conforming to a different metamodel based on a certain set of rules (Wimmer, Perez, Jouault, & Cabot, 2012). An example of an MMT performed by EMF is the transformation of an Ecore model into a UML model. This process involves extracting the entities and attributes and defining them as classes and properties, and including references and associations to create the UML model.

Automated Code Generation from a Scenario Model

Once a conceptual model has been created, a conceptual scenario can be defined by running the metamodel and describing the attributes of all entities in this instance of the model. For ASDL, this is performed by executing the automated validation process included within Eclipse to ensure that all classes and attributes have defined requirements for the expected valid data. These standards are included within the metamodel and mainly describe items such as ensuring that all attributes have a data type, and all class relationships define the expected cardinality. The model, editing and testing codes are automatically obtained by using EMF's in-built code generation tools. This automated code has a separate Java class for each class in the conceptual model, which includes the getter and setter methods for all attributes. It is possible to allow for changes to be made in the generation of code for each class as necessary based on the required behavior. Once these classes have been generated, the metamodel is considered complete and an XML schema is generated and validated. A model of a conceptual scenario can now be defined using the metamodel. In this next step, the Eclipse Model Editor is used to define the operational scenario and an XML script is created. Eclipse automatically generates an XML Metadata Interchange (XMI) file, which is a specialized application of XML and is used to represent the model.

Methodology 1: ASDL-Ecore Scenario Modeling

To provide an easy-to-use drag and drop framework to construct ASDL scenario models, a recent effort presented a graphical modeling and editing interface to ASDL (Jafer et al., 2017a). The graphical scenario specification tool is developed using EMF Forms within EMF which provides a rapid mechanism to develop tools for modeling languages. Figure 8 illustrates the graphical interface used to quickly specify an aircraft model in ASDL.



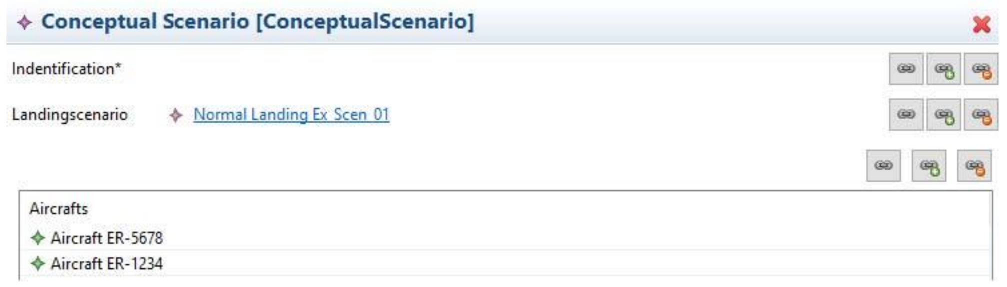
Call Sign*	ER-1234		
Pilot*	Pilot Jane Smith		
Origin*	Airport ATL	Destination*	Airport DAB
Departing Runway*	Runway 27R	Landing Runway*	Runway 7L
		Atc	ATC Jane Doe
Flightproperties*	Flight Properties	Weather Properties*	Weather
State	Cruise		

Figure 8. Aircraft modeling GUI. Adapted from “Graphical Specification of Fight Scenarios with Aviation Scenario Definition Language (asdl)” by S. Jafer et al., 2017a, *AIAA Modeling and Simulation Technologies Conference.*

Ultimately, for the given landing scenario discussed previously, all entities such as weather, pilot, ATC, runway, etc. can be quickly specified using EMF-supported UI. The overall scenario created using this approach is illustrated in Figure 9, showing aircraft entity, followed by all other entities (hidden to preserve space).

Methodology 2: System Entity Structure

The system theory-based approach to modeling and simulation has resulted in many enhancements in the field, one of which is System Entity Structure (SES) (Ören & Zeigler, 2012). SES is a high-level ontology which was introduced for knowledge representation of decomposition, taxonomy and coupling of systems (Kim, Lee, Christensen, & Zeigler, 1990). SES is a useful ontological framework to define data engineering ontologies.



Conceptual Scenario [ConceptualScenario]

Indentification*	
Landingscenario	Normal Landing Ex Scen 01

Aircrafts
- Aircraft ER-5678
- Aircraft ER-1234

Figure 9. Overall landing scenario specified graphically in EMF. Adapted from “Graphical Specification of Fight Scenarios with Aviation Scenario

Definition Language (asdl)" by S. Jafer et al., 2017a, *AIAA Modeling and Simulation Technologies Conference.*

SES enables fundamental representation of hierarchical modular model providing a design space via the elements of a system and their relationships in hierarchical and axiomatic manner. SES is a declarative knowledge representation scheme that characterizes the structure of a family of models in terms of decompositions, component taxonomies, and coupling specifications and constraints (Zeigler, 1984). As it has been described in a number of publications (Pawletta et al., 2016; Zeigler & Hammonds, 2007), SES supports development, pruning, and generation of a family of hierarchical simulation models. SES is a formal ontology framework, axiomatically defined, to represent the elements of a system (or world) and their relationships in hierarchical manner. Figure 10, created by the author, provides a quick overview of the nodes and relationship involved in a SES. Entities represent things that have existence in a certain domain. They can have variables which can be assigned a value within given range and types. An Aspect expresses a way of decomposing an object into more detailed parts and is a labeled decomposition relation between the parent and the children. Multi-Aspects are aspects for which the components are all of the one kind. A Specialization represents a category or family of specific forms that a thing can assume. It is a labeled relation that expresses alternative choices that a system entity can take on.

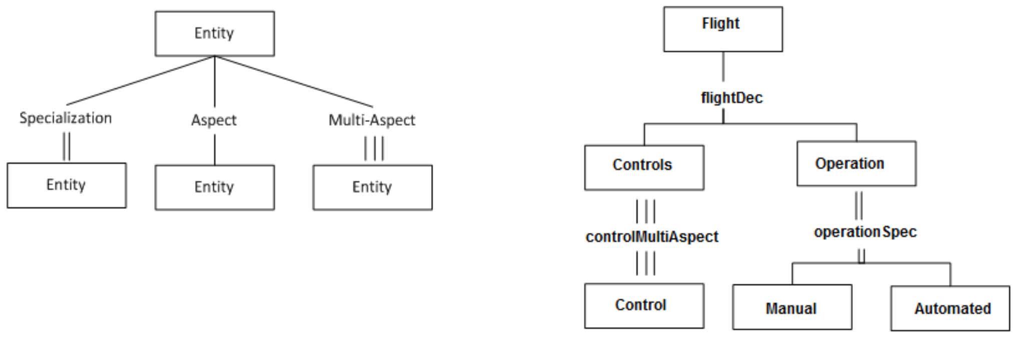


Figure 10. Nodes and relationship involved in a SES.

Given an SES tree, when suitably pruned, SES provides specific SES instantiations for investigation and analysis. Pruning is defined as assigning the values to the variables and resolving the choices in Aspect, Multi-Aspect and Specialization relations. While there may be several Aspect nodes for several decompositions of the system on the same hierarchical level, a particular subset can be chosen in pruning based on the purpose. Specializations enable to capture various variants of an entity, one which needs to be selected during pruning. The cardinality in Multi-Aspect relations is also specified in pruning, resulting in the Pruned Entity Structure (PES), which is a selection-free tree.

SES Metamodel

The SES metamodel captures all SES constructs and their relationships. Figure 11, created by the author, presents an overview of a representative SES metamodel that has been developed using ECORE.

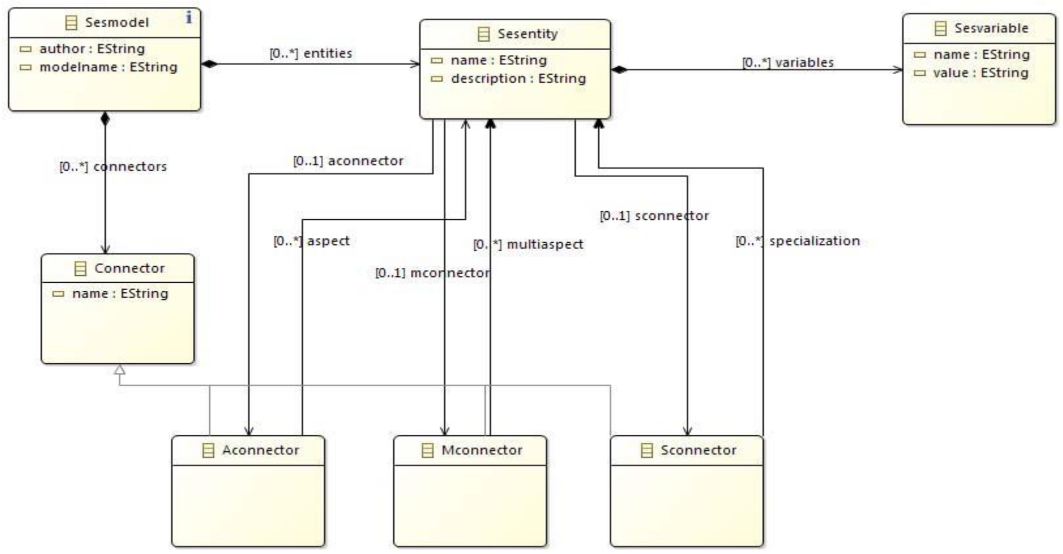


Figure 11. System Entity Structure metamodel.

The constructs of the metamodel are Entity, Specialization, Aspect, MultiAspect and Attribute classes. An Attribute has a name and value field which are specified as EString type for the sake of simplicity. In order to exemplify the relationships between the constructs, we can have a look at the unidirectional references between Entity and MultiAspect. An Entity type node can have a zero to n multi-aspect to MultipleAspect type node, and further a MultipleAspect type node has a reference to an Entity type nodes. The process followed for developing the metamodel and a specific scenario can be seen in the author-created Figure 12.

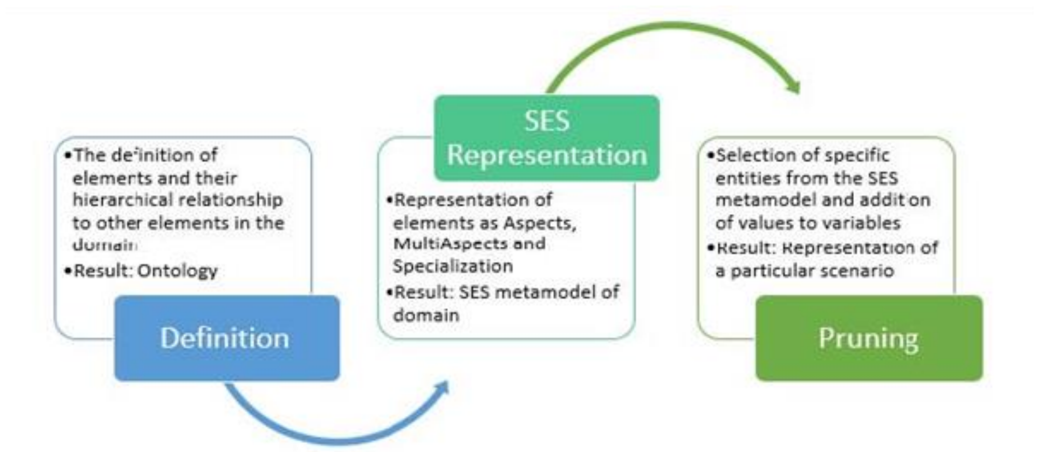

•The definition of elements and their hierarchical relationship to other elements in the domain
•Result: Ontology
Definition
SES Representation
•Representation of elements as Aspects, MultiAspects and Specialization
•Result: SES metamodel of domain
•Selection of specific entities from the SES metamodel and addition of values to variables
•Result: Representation of a particular scenario
Pruning

Figure 12. Process followed for developing the metamodel and a specific scenario.

ASDL Metamodel in SES

SES-based metamodeling approach (Zeigler & Sarjoughian, 2013) provides a high-level ontology for knowledge representation of decomposition, taxonomy and coupling of ASDL scenarios. The ASDL Metamodel captures all possible meta elements of a flight operation scenario (landing, reroute, departure) via a SES. A representative excerpt of ASDL Metamodel is presented in Figure 13, created by the author. The top-level Scenario entity is decomposed using the scenarioDec aspect node into Environment, Entities and Events. entityMultiAsp multi-aspect node decomposes Entities to multiple nodes Entity. entitySpec specialization node is then used to capture the different types of Entity. Three examples from a larger set that are depicted in the figure are Aircraft, Airports which is the declaration for multiple Airport, and Weather. aircraftDec is then used to identify the aspects of an Aircraft that are of interest as elements of a scenario. These are namely Flight and Pilot. Flight is then decomposed into its states: Position, Attitude, Angular Velocity and Translational Velocity. Airport is decomposed into ATCs and Runways which are declaration for multiple ATC and Runway. WeatherStateDec decomposes Weather into Wind and Temperature. eventDec decomposes an event into a Guard and an Action. Two Guard types are State and Time. eventSpec is on the other side used to capture various types of Event. Examples are Reroute, Landing and Departure. Finally, a Landing event can be specified as either NormalLanding, CrosswindLanding, or ShortFieldLanding.

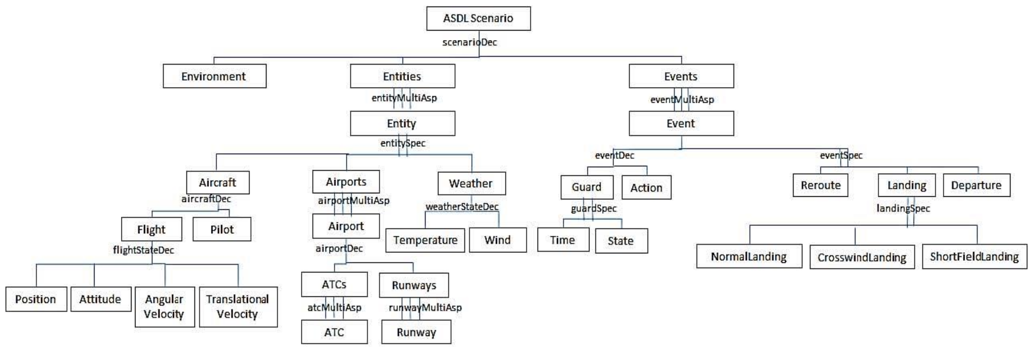


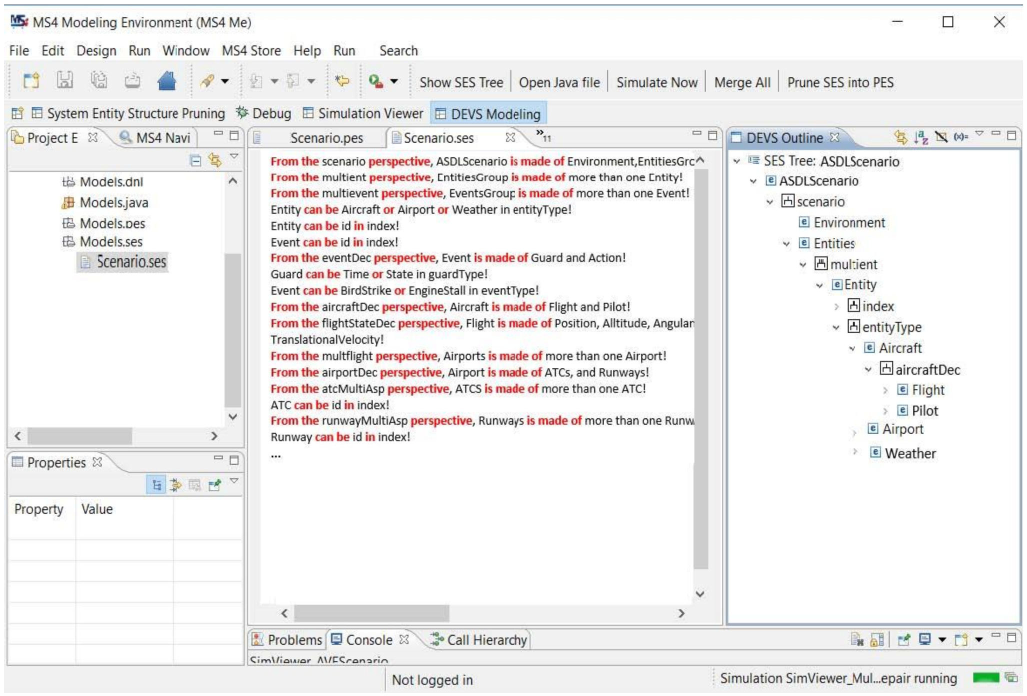
Figure 13. ADL metamodel extract.

In an SES tree, attributes are leaf entities that capture scenario's data values. As an example, Position entity has attributes Latitude, Longitude and Altitude. Default values for attributes can be easily set at metamodeling. Such data could appear on SES tree for quick reference. While the presented ASDL SES hierarchy excerpt is not complete, it includes enough number of elements to be representative as a metamodel that captures various possible scenario elements. The complete metamodel captures all the possible scenario elements that are available in ASDL. SES structure can also be implemented in the MS4 Me, which is an Eclipse-based tool suite that provides a quick development environment to specify SES entities and their relationships. Figure 14, created by the author, is the partial ASDL implementation in MS4 Me environment with SES tree constructed on the right-side section.

ASDL-SES Scenario Modeling

With the given SES tree in Figure 13 consisting of all possible elements of the simulation scenario, the scenario modeling activity is as simple as Pruning of this tree to hand pick a very particular scenario. Values are assigned to attributes, and selections are performed for Aspect, Multi-aspect, and Specialization. The resulting selection-free tree is the model representation of that particular scenario. See Figure 15, created by the author.

Pruning can be conducted via automated means using a scripting front-end that sets the attributes values and selections in decision nodes such as cardinalities at multi-aspect nodes or types at specialization nodes. The pruning procedure resulted in Figure 15, however was accomplished manually. The ASDL Scenario Metamodel is used to construct a modeling toolbox that is composed of decision-free nodes of the SES tree. As the user adds the Scenario to its mode, Environment appears a decision-free elect of the scenarioDec. Then the user selects which and how many Entities will be added. Referring to the operational scenario of Normal Landing provided in the Simulation Scenario Development subsection, the user proceeds to Step 2 by adding the NormalLanding event to the model. In Step 3 and 4, the user then adds the Aircraft and Airport entities. Step 5 follows by adding Weather element and finally the last step is when the user specifies the missing values of selected attributes (gray boxes), specifying all the details according to the operational scenario.


MS4 Modeling Environment (MS4 Me)
File Edit Design Run Window MS4 Store Help Run Search
Show SES Tree | Open Java file | Simulate Now | Merge All | Prune SES into PES
System Entity Structure Pruning | Debug | Simulation Viewer | DEVS Modeling
Project E | MS4 Navi
Models.dnl
Models.java
Models.pes
Models.ses
Scenario.ses
Properties
Property | Value
Scenario.pes | Scenario.ses
DEVS Outline
SES Tree: ASDLScenario
ASDLScenario
scenario
Environment
Entities
multient
Entity
index
entityType
Aircraft
aircraftDec
Flight
Pilot
Airport
Weather
Problems | Console | Call Hierarchy
Not logged in
Simulation SimViewer_Mul...epair running

```
From the scenario perspective, ASDLScenario is made of Environment,EntitiesGrc
From the multient perspective, EntitiesGroup is made of more than one Entity!
From the multievent perspective, EventsGroup is made of more than one Event!
Entity can be Aircraft or Airport or Weather in entityType!
Entity can be id in index!
Event can be id in index!
From the eventDec perspective, Event is made of Guard and Action!
Guard can be Time or State in guardType!
Event can be BirdStrike or EngineStall in eventType!
From the aircraftDec perspective, Aircraft is made of Flight and Pilot!
From the flightStateDec perspective, Flight is made of Position, Alltitude, Angular
TranslationalVelocity!
From the multflight perspective, Airports is made of more than one Airport!
From the airportDec perspective, Airport is made of ATCs, and Runways!
From the atcMultiAsp perspective, ATCS is made of more than one ATC!
ATC can be id in index!
From the runwayMultiAsp perspective, Runways is made of more than one Runw
Runway can be id in index!
...
```

Figure 14. Representation of ASDL scenario metamodel in SES.

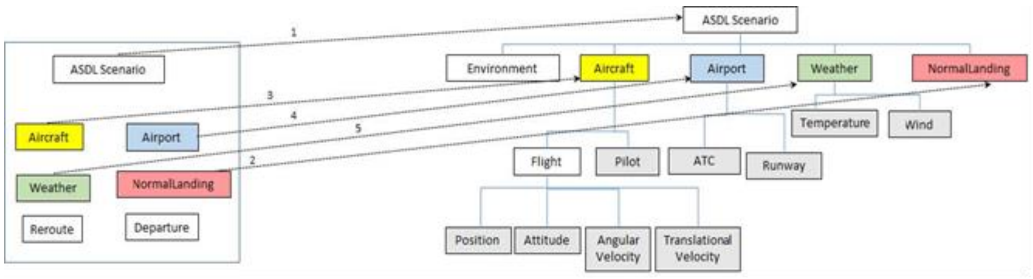


Figure 15. Scenario development with pruning.

Ultimately, the MS4 Me tool suite can be utilized to work from any such pruning to generate the complete space by enumeration or it can sample from this space randomly. Automated pruning in the form of enumerative and random pruning are under consideration by MS4 Me team. Enumerative pruning can generate completely pruned entity structures sequentially assuring that each family member is produced once and only once (Zeigler, Kim, & Praehofer, 2000). This "brute force" method is sufficient for relatively small solution spaces – recall that the family size grows geometrically with number of choices. Random pruning samples from the family of PES will make choices with uniform probability wherever the pruning script has not given the pruner a basis for decision. By starting from a different initial seed for its pseudo-random number generator at each iteration, the pruner draws different random samples from the solution space. This process will be constrained by a set of rules.

Furthermore, MS4 Me can develop rules that direct the pruning process as well the pre-and post-processing of the pruning results. Such rules will exploit partial contexts to enable a rule to be applied to every occurrence of an entity that satisfies a partial context. In general, there may be more than one rule and concomitantly, more than one partial context may apply to an entity. Accordingly, the developed algorithms will be enabled to make decisions in which selections are ambiguous. For example, one approach is to order partial contexts by length, longest first - on the basis that longer paths are more specific than shorter paths. For each entity occurrence that it encounters, the algorithm finds the longest (most specific) partial context that matches the occurrence under consideration and applies the associated rule to it. Conditional rules can be developed in which choices made in one location of the structure will condition those in other locations.

Comparison of Approaches

Both approaches require an understanding of the subject domain in order to create a metamodel. The definitions and relationships between these elements need to be understood so they can be represented hierarchically within the ontology. The SES approach requires a more strict and formal definition of the ontology since MultiAspects and Specializations are separated from other Aspects, which are the other child elements (Jafer, Chhaya, Updegrove, & Durak, 2018). On the other hand, an OWL ontology requires metamodeling using Ecore or another framework to extract an XML schema and generate a specific scenario, whereas an SES Editor such as MS4 Me can generate the XML directly from a pruned SES model. The additional step of translating an OWL file into Ecore before generating a scenario makes the use of SES more favorable in the authors' eyes.

Application

One of the applications of our proposed ontology-driven scenario development is in the domain of aviation training. Targeting the FAA Academy Air Traffic Control program, we are building a scenario-based training environment that enhances controller's training by providing a practice environment where trainees investigate various air traffic scenarios (Updegrove & Jafer, 2017). Based on the concept of scenario specification and modeling adopted in ASDL, we are developing a GUI-based environment where controller instructors specify air traffic scenario metrics and properties, conduct performance and evaluation studies, and monitor trainee's responses to gauge the learning process. On the other hand, trainees are provided with a close-to-reality simulation environment, where they practice various scenarios by analyzing and reacting to the scenario events and making optimal choices in controlling air traffic events (Chhaya, Jafer, Coyne, Thigpen, & Durak, 2018).

Similarly, the defense domain can significantly benefit from scenario-based simulation technologies for training, guidance, and decision support purposes. The MSDL (Military Scenario Definition Language Product Development) was proposed for this purpose and have been widely used in the defense domain (SISO, 2008).

The application of ASDL has been described for the following domains in other works: generation of flight simulation scenarios (Jafer, Chhaya, Durak, & Gerlach, 2018), scenario-based challenges for Next Generation Aviation Technology (Moallemi, Jafer, & Chhaya, 2018) and enhancing scenario-centric ATC training (Chhaya et al., 2018).

Conclusion

This paper presents a model-based scenario development approach that exploits Eclipse Modeling Framework (EMF) Ecore and System Entity Structure (SES) for metamodeling and modeling. Despite its key role in a simulation study, there is no structured and well-formed methodology for scenario development. By presenting two distinct metamodeling approaches for a Domain Specific Language (DSL) recently published for aviation simulation scenario specification (Aviation Scenario Definition Language – ASDL), ontology development and model-based scenario specification stages are presented. EMF uses the Ecore format, which is a subset of UML class diagrams to describe entities and their relationships. SES represents the elements of a system and their relationships in a hierarchical manner. Given the similar structure that both Ecore and SES follow, it is not easy to draw a fine comparison line between the two methodologies. Obviously, through steps of model transformations, one can easily, and even automatically, translate a DSL metamodel from one approach to another. Hierarchical structure and entity specification are among the main concepts shared by Ecore and SES in constructing a DSL ontology. The key is the selection of scenario-specific entities from a DSL ontology, which has been evidently made easy by tool support, providing modelers various means of automation in specifying, validating, and verifying a scenario. This work showcases the capabilities of both EMF and SES as metamodel frameworks for scenario-based modeling, but it can be extended to investigate the full tool suites available in each platform to determine its suitability for all aspects of the modeling process. The research demonstrated in this article has already been utilized to develop a scenario-based training tool for air traffic controllers at the FAA Academy. Moving the effort through standardization is already underway, where XML representation of SES already exists and ASDL Ecore/XML is currently being researched. Automated model checking and transformation as well as more rigorous tool support are two future directions in this area, both for Ecore and SES.

References

Alatrish, E. (2013). Comparison some of ontology. *Journal of Management Information Systems, 8*(2), 18-24.

ASDL Ontology. (2016). Retrieved from https://github.com/ASDL-prj/Ontology

Bechhofer, S. (2009). Owl: Web ontology language. In *Encyclopedia of database systems* (pp. 2008–2009). Retrieved from https://link.springer.com/ referenceworkentry/ 10.1007%2F978-0-387-39940-9_1073

Brambilla, M., Cabot, J., & Wimmer, M. (2012). Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, *1*(1), 1-182.

Bruneliere, H., Cabot, J., Jouault, F., & Madiot, F. (2010). Modisco: A generic and extensible framework for model driven reverse engineering. In *Proceedings of the ieee/acm international conference on automated software engineering* (pp. 173–174). Antwerp, Belgium.

Budinsky, F., Steinberg, D., Ellersick, R., Grose, T. J., & Merks, E. (2004). *Eclipse modeling framework: A developer's guide*. Addison-Wesley.

Ceh, I., Crepinšek, M., Kosar, T., & Mernik, M. (2011). Ontology driven development of domain-specific languages. *Computer Science and Information Systems, 8*(2), 317–342.

Chandrasekaran, B., Josephson, J. R., & Benjamins, V. R. (1999). What are ontologies, and why do we need them? *IEEE Intelligent Systems and their applications*, *14*(1), 20–26.

Chhaya, B., Jafer, S., Coyne, W. B., Thigpen, N. C., & Durak, U. (2018). Enhancing scenario-centric air traffic control training. In *2018 AIAA modeling and simulation technologies conference* (p. 1399). Kissimmee, FL.

Department of Defense. (1998, January). *DoD modeling and simulation (M&S) glossary*. Washington, DC: Author.

Durak, U., Schmidt, A., & Pawletta, T. (2015). Model-based testing for objective fidelity evaluation of engineering and research flight simulators. In *AIAA modeling and simulation technologies conference* (p. 2948). Kissimmee, FL.

Durak, U., Topçu, O., Siegfried, R., & Oguztuzun, H. (2014). Scenario development: A model-driven engineering perspective. In *2014* *international conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)* (pp. 117–124). Vienna, Austria.

FAA Flight Standards Service AFS Flight Program Division. (2012). *AFS flight program flight operations manual*. Retrieved from http://fsims.faa.gov/wdocs/other/fom.htm

Farquhar, A., Fikes, R., & Rice, J. (1997). The ontolingua server: A tool for collaborative ontology construction. *International Journal of Human-Computer Studies*, *46*(6), 707-727.

Fowler, M. (2010). *Domain-specific languages*. New York, NY: Pearson Education.

Gronback, R. (2014). *Eclipse modeling framework (emf).* Retrieved from https://eclipse.org/ modeling/ emf/

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, *5*(2), 199–220.

Hilera, J. R., & Fernández-Sanz, L. (2010). Developing domain-ontologies to improve software engineering knowledge. In *Software engineering advances (icsea), 2010 fifth international conference on* (pp. 380–383). Nice, France.

Hudak, P. (1997). Domain-specific languages. *Handbook of Programming Languages*, *3*(39-60), 21. Retrieved from https://pdfs.semanticscholar.org/ b06c/ 06de5335a0e53ad7122419886890c2cab2a4.pdf

Jafer, S., Chhaya, B., & Durak, U. (2017a). Graphical specification of flight scenarios with aviation scenario definition language (asdl). In *AIAA modeling and simulation technologies conference* (p. 1311). Grapevine, TX.

Jafer, S., Chhaya, B., & Durak, U. (2017b). Owl ontology to Ecore metamodel transformation for designing a domain specific language to develop aviation scenarios. In *Proceedings of the symposium on model-driven approaches for simulation engineering* (p. 3). Grapevine, TX.

Jafer, S., Chhaya, B., Durak, U., & Gerlach, T. (2016). Formal scenario definition language for aviation: aircraft landing case study. In *AIAA modeling and simulation technologies conference* (p. 3521). Washington, D.C.

Jafer, S., Chhaya, B., Durak, U., & Gerlach, T. (2018). Automatic generation of flight simulation scenarios with aviation scenario definition language. *Journal of Aerospace Information Systems*, *15*(4), 193-202.

Jafer, S., Chhaya, B., Updegrove, J., & Durak, U. (2018). Schema-based ontological representations of a domain-specific scenario modeling language. *Journal of Simulation Engineering*, *1*.

Kim, T.-G., Lee, C., Christensen, E. R., & Zeigler, B. P. (1990). System entity structuring and model base management. *IEEE Transactions on Systems Man and Cybernetics*, *20*(5), 1013–1024.

Maedche, A., & Staab, S. (2001). Ontology learning for the semantic web. *IEEE Intelligent systems*, *16*(2), 72-79.

McGuinness, D. L. (2004). Owl web ontology language overview. *W3C recommendation*, *10*(10).

Moallemi, M., Jafer, S., & Chhaya, B. (2018). Scenario specification challenges for next generation aviation technology demonstrations. In *2018 AIAA modeling and simulation technologies conference* (p. 1396). Kissimmee, FL.

Musen, M. A. (2015). The protégé project: A look back and a look forward. *AI matters*, *1*(4), 4-12.

North Atlantic Treaty Organization. (2015, January). *Guideline on scenario development for (distributed) simulation environments.* Retrieved from https://www.sto.nato.int/ publications/STO%20Technical%20Reports/ STO-TR-MSG-086-Part-II/$$TR-MSG-086-Part-II-ALL.pdf

Noy, N. F., & McGuinness, D. L. (2001). *Ontology development 101: A guide to creating your first ontology*. Retrieved from https://protege.stanford.edu/ publications/ ontology_development/ontology101.pdf

Ören, T. I., & Zeigler, B. P. (2012). System theoretic foundations of modeling and simulation: a historic perspective and the legacy of a Wayne Wymore. *Simulation*, *88*(9), 1033–1046.

Pan, J. Z., Staab, S., Aßmann, U., Ebert, J., & Zhao, Y. (2012). *Ontology-driven software development*. New York, NY: Springer.

Pawletta, T., Schmidt, A., Zeigler, B. P., & Durak, U. (2016). Extended variability modeling using system entity structure ontology within matlab/simulink. In *Proceedings of the 49th annual simulation symposium* (p. 22). Pasadena, CA.

Saeki, M., & Kaiya, H. (2006). On relationships among models, meta models and ontologies. In *Proceedings of the proceedings of the 6th oopsla workshop on domain-specific modeling (dsm 2006).* Portland, OR. doi:10.1.1.103.6720

Schmidt, A., Durak, U., & Pawletta, T. (2016). Model-based testing methodology using system entity structures for matlab/simulink models. *Simulation, 92*(8), 729–746.

SESAR. (2015). *Welcome to the Sesar integrated dictionary*. Retrieved from https://ext.eurocontrol.int/lexicon/index.php/SESAR

Siegfried, R., Laux, A., Rother, M., Steinkamp, D., Herrmann, G., Lüthi, J., & Hahn, M. (2012). *Scenarios in military (distributed) simulation environments*. Retrieved from https://www.sto.nato.int/publications/ STO%20Technical%20Reports/STO-TR-MSG-086-Part-II/$$TR-MSG-086-Part-II-ALL.pdf

Siegfried, R., Oguztüzün, H., Durak, U., Hatip, A., Herrmann, G., Gustavson, P., & Hahn, M. (2013). *Specification and documentation of conceptual scenarios using base object models (boms)*. Retrieved from https://www.researchgate.net/publication/ 256939472_ Specification_and_ Documentation_of_Conceptual_Scenarios_Using_Base_Object_Models_ BOMs

Simulation Interoperability Standards Organization. (2008, October). Standard for: Military scenario definition language (msdl). Retrieved from https://www.sisostds.org/ DigitalLibrary.aspx?Command=Core_ Download&EntryId=30830

Skrypuch, N. (2007). *Model to text (m2t).* Retrieved from http://www.eclipse.org/modeling/m2t/

SISO Base Object Model Product Development Group. (2006, March). *Base object model (bom) template specification.* Retrieved from https://www.sisostds.org/DesktopModules/ Bring2mind/DMX/API/ Entries/Download?Command=Core_Download&EntryId=30820&PortalId =0&TabId=105

Stanford Center for Biomedical Informatics Research. (2017). *A free, open-source ontology editor and framework for building intelligent systems.* Retrieved from http://protege.stanford.edu/

Steinberg, D., Budinsky, F., Merks, E., & Paternostro, M. (2008). *Emf: Eclipse modeling framework*. New York, NY: Pearson Education.

Topçu, O., Durak, U., O˘guztüzün, H., & Yilmaz, L. (2016). *Distributed simulation: A model driven engineering approach*. New York, NY: Springer.

Updegrove, J., & Jafer, S. (2017). Recommendations for next generation air traffic control training. In *Digital avionics systems conference (dasc), 2017 IEEE/AIAA 36th* (pp. 1–6). St. Petersburg, FL.

Wimmer, M., Perez, S. M., Jouault, F., & Cabot, J. (2012). A catalogue of refactorings for model-to-model transformations. *Journal of Object Technology*, *11*(2), 2-1.

Wittman Jr, R. L. (2009). Defining a standard: The military scenario definition language version 1.0 standard. In *Proceedings of the 2009 spring simulation multiconference* (p. 73). San Diego, CA.

Zeigler, B. P. (1984). *Multifacetted modelling and discrete event simulation*. San Diego, CA: Academic.

Zeigler, B. P., & Hammonds, P. E. (2007). *Modeling and simulation-based data engineering: introducing pragmatics into ontologies for net-centric information exchange*. Amsterdam, Netherlands: Elsevier.

Zeigler, B. P., Kim, T. G., & Praehofer, H. (2000). *Theory of modeling and simulation*. San Diego, CA: Academic.

Zeigler, B. P., & Sarjoughian, H. S. (2013). *Guide to modeling and simulation of systems of systems.* New York, NY: Springer.