Scenario-Based Development and Verification of Domain-Specific Languages

Bharvi Chhaya

Follow this and additional works at: https://commons.erau.edu/edt

Part of the Electrical and Computer Engineering Commons

Scholarly Commons Citation
https://commons.erau.edu/edt/550

This Dissertation - Open Access is brought to you for free and open access by Scholarly Commons. It has been accepted for inclusion in PhD Dissertations and Master's Theses by an authorized administrator of Scholarly Commons. For more information, please contact commons@erau.edu.
SCENARIO-BASED DEVELOPMENT AND VERIFICATION OF DOMAIN-SPECIFIC LANGUAGES

by Bharvi Chhaya

This dissertation was prepared under the direction of the candidate’s Dissertation Committee Chair, Dr. Shafagh Jafer, and has been approved by the members of the dissertation committee. It was submitted to the College of Engineering and accepted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Electrical Engineering and Computer Science.
I would like to thank my advisor, Dr. Shafagh Jafer, without whose guidance and support this dissertation would not be complete. I appreciate the advice and encouragement of my committee and the department leadership who have supported me through the years. I want to recognize Dr. Keith Garfield, Dr. Massood Towhidnejad, Dr. Omar Ochoa and Dr. Dan Macchiarella from my committee, and Dr. Tim Wilson, Dr. Remzi Seker, Dr. Radu Babiceanu and Professor Farahzad Behi from the EECS Department.

I want to thank everyone I have worked with during my time at ERAU. I really appreciate the guidance and leadership of Dr. Mohammad Moallemi and the support of my friends and colleagues, Chris Shannon and Jessica Updegrove.

I would like to acknowledge my roommate Dr. Shibani Bhatt for her help and encouragement in all aspects during my stay here. I want to thank Himanish Ganjoo who volunteered to help me with my research when I was stuck and was a source of constant support during the process. I am grateful to Ravi Shrivastava and Yash Mehta, who were always there for me during this period.

Finally, I would like to thank my father, Dr. Nimish Chhaya, has been a steady hand of guidance and an immovable rock of support. I would never be who I am without him, and for all that he has been and done for me, I am immensely grateful. My mother, Mrs. Parul Chhaya, is the reason I could follow my dreams. She pushed me to achieve what I have, and to be the best version of myself I could be. I have been inspired by her strength and resolve, seeking to emulate it in my own struggles. Her unwavering support has been instrumental to my success, and I cannot thank her enough.
Table of Contents

Acknowledgments.............................................................................................................. iii

Table of Contents ............................................................................................................... iv

List of Tables .................................................................................................................... vii

List of Figures .................................................................................................................... ix

Abstract .............................................................................................................................. xi

Introduction ......................................................................................................................... 1

1.1 Motivation ...................................................................................................................... 2

1.2 Challenges .................................................................................................................... 3

1.3 Research Questions .................................................................................................... 4

1.4 Contribution ................................................................................................................. 6

1.5 Roadmap ....................................................................................................................... 7

Literature Review ................................................................................................................ 9

2.1 Domain-Specific Languages ....................................................................................... 9

2.2 Scenario-Based Development .................................................................................... 15

2.3 DSL Evaluation and Verification ............................................................................... 24

Aviation Scenario Definition Language ........................................................................... 29

3.1 Background .................................................................................................................. 29

3.2 Approach Methodology: Ontology ............................................................................. 30

3.3 Model Design and Behavior ....................................................................................... 32
8.1 Conclusions................................................................. 126

8.2 Recommendations for Future Work...................................... 131

References................................................................................. 135
# List of Tables

Table 1: List of entities and values identified ............................................................... 53
Table 2: List of entities and interactions ........................................................................ 54
Table 3: Elements in the domain model based on the scenario .................................... 56
Table 4: List of actions and states identified from Scenario NL01 ............................... 60
Table 5: Comparison of DSL modeling approaches ..................................................... 67
Table 6: Entities and values obtained from Scenario 1 by the user .............................. 71
Table 7: Entities and interactions obtained from Scenario 1 by the user ..................... 72
Table 8: Elements in the domain model based on Scenario 1 by the user .................... 73
Table 9: Entities and values obtained from Scenario 1 by the author .......................... 74
Table 10: Entities and interactions obtained from Scenario 1 by the author ............... 75
Table 11: Elements in the domain model based on Scenario 1 by the author .............. 76
Table 12: Entities and values obtained from Scenario 2 by the user .......................... 78
Table 13: Entities and values obtained from Scenario 2 by the author ....................... 79
Table 14: Entities and interactions obtained from Scenario 2 by the user ................... 80
Table 15: Entities and interactions obtained from Scenario 2 by the author ............... 81
Table 16: Elements in the domain model based on scenario 2 by the user ................. 83
Table 17: Elements in the domain model based on scenario 2 by the author .............. 84
Table 18: Differences in the two domain models ....................................................... 92
Table 19: Time log filled out by user for Scenario 1 .................................................... 93
Table 20: Snapshot of NASA-TLX responses from the user for scenario parsing using the DoSS framework ................................................................................. 98
Table 21: Frequency of words obtained from the first scenario ................................. 117
Table 22: Frequency of nouns obtained from 10 scenarios ............................................ 118
Table 23: Frequency of nouns obtained from 30 scenarios from three sources .......... 119
Table 24: Relationships between elements identified from scenarios ......................... 121
List of Figures

Figure 1: Types of scenarios ........................................................................................................ 18
Figure 2: BOM composition ........................................................................................................ 34
Figure 3: Top-level view of ConceptualScenario in BOM metamodel ........................................ 35
Figure 4: Typical scenario development process ....................................................................... 37
Figure 5: Properties of the Aircraft class .................................................................................. 39
Figure 6: Properties of (a) ConceptualScenario, and (b) ScenarioIdentification classes .. 41
Figure 7: Process for creating a domain model .......................................................................... 47
Figure 8: Steps outlining the DoSS framework ........................................................................... 47
Figure 9: Elements of Aircraft class in ontology ...................................................................... 50
Figure 10: Domain model for aircraft landing in EMF ............................................................. 51
Figure 11: Representation of Aircraft entity model ................................................................. 57
Figure 12: Representation of Airport entity model ................................................................... 57
Figure 13: Representation of Runway entity model .................................................................. 57
Figure 14: Representation of ATC model ................................................................................. 58
Figure 15: Representation of Pilot entity model ....................................................................... 58
Figure 16: Representation of Clearance entity model ............................................................ 58
Figure 17: Combined domain model based on Scenario NL01 ............................................... 59
Figure 18: Preliminary domain model based on scenarios ....................................................... 59
Figure 19: Preliminary statechart based on Scenario NL01 .................................................... 61
Figure 20: Domain model in EMF using the scenario-based approach .................................... 64
Figure 21: Statechart for aircraft landing based on scenarios ................................................. 65
Figure 22: Domain model based on Scenario 1 by the user ..................................................... 74
Figure 23: Domain model based on Scenario 1 by the author .......................................... 77
Figure 24: Domain model based on Scenario 2 by the user ........................................... 85
Figure 25: Domain model based on Scenario 2 by the author .......................................... 86
Figure 26: Domain model based on all scenarios by the user ........................................... 87
Figure 27: Domain model based on all scenarios by the author ........................................... 87
Figure 28: All top-level entities for the two domain models ............................................ 89
Figure 29: Attributes and references for the two domain models – 1/5 ............................ 90
Figure 30: Attributes and references for the two domain models – 2/5 ............................ 90
Figure 31: Attributes and references for the two domain models – 3/5 ............................ 91
Figure 32: Attributes and references for the two domain models – 4/5 ............................ 91
Figure 33: Attributes and references for the two domain models – 5/5 ............................ 92
Figure 34: Time taken by the user to parse each scenario using the DoSS framework .... 94
Figure 35: NASA Task Load Index .................................................................................. 97
Figure 36: Overall NASA-TLX scores for effort selected by the user ............................. 99
Figure 37: Statechart based on all scenarios by the user ................................................ 103
Figure 38: Elements added to en route ontology ............................................................ 112
Figure 39: Development process for ASDL scenario generator ..................................... 113
Figure 40: Snippet of XML script for the sample scenario ............................................ 114
Figure 41: High-level view of scenario-based aviation ontology ................................. 122
Figure 42: Components of airspace, airport, and flight class in ontology ................. 122
Figure 43: Comparison of high-level ASDL and scenario-driven ontologies ............ 123
Figure 44: Comparison of components in ASDL and scenario-driven ontologies .... 124
Abstract

The use of domain-specific languages (DSLs) has increased manifold for problem-solving in specific domain areas as they allow for a wider variety of expressions within their domain. Modeling using DSLs has shown high increases in productivity after accounting for the time and cost expended in developing them, making them a suitable target for improvement in order to reap higher rewards. The currently used approach for domain modeling involves the creation of an ontology which is then used to describe the domain model. This ontology encapsulates all domain knowledge and can be cumbersome to create, requiring external sources of information and assistance from a domain expert.

This dissertation first discusses the use and importance of DSLs for scenario generation for a domain and presents an extension to the Aviation Scenario Definition Language (ASDL). The main contribution of this dissertation is a novel framework for scenario-based development of DSLs, called the Domain-Specific Scenario (DoSS) framework. This framework proposes the use of scenarios in natural language, which are currently used in requirements engineering and testing, as the basis for developing the domain model iteratively. An example of the use of this approach is provided by developing a domain model for ASDL and comparing the published model with one obtained using DoSS. This approach is supplemented with a case study to validate the claim that DoSS is easier to use by non-experts in the domain by having a user create a model and comparing it to one obtained by the author. These models were found to be almost identical, showing a promising return for this approach. The time taken and effort required to create this model by the user were recorded and found to be quite low,
although no similar results have been published so no comparison could be made.

Statecharts are then used for verification of scenarios to ensure the conformity between scenarios and models. The dissertation also discusses applications of the ideas presented here, specifically, the use of ASDL for Air Traffic Control training scenarios and the use of DoSS for ontology generation.
Chapter 1

Introduction

Domain-specific languages (DSLs) are computer languages tailored to a specific application domain. As a result, DSLs are often more expressive for that particular domain, offering ease of use. This allows domain experts, who may not be familiar with programming and general-purpose programming languages, to use a DSL to express ideas and concepts in their domain, which is commonly not possible otherwise. Some examples of DSLs are SQL for querying relational databases, Mathematica for symbolic mathematics, and HTML for web pages. A program or specification written in a DSL can then be interpreted or compiled into a General-Purpose Language (GPL). In other cases, the specification can represent simple data that is processed by other systems (Bettini, 2016).

DSLs often start simple, based on an initially limited understanding of the domain, but then grow more and more sophisticated over time (Hudak, 1996). There are four key elements to the definition of a DSL (Fowler, 2010):

- A DSL is a **computer language**. While it is easier for humans to understand, it must be executable by a computer. It differs from a programming language in that it does not need to be Turing complete (Sun, Demirezen, Mernik, Gray, & Bryant, 2008).
• A DSL must be a **language by nature**. It should have a sense of fluency where there are individual expressions, but they can also be composed together.

• A DSL must only have **limited expressiveness**. It must only have a bare minimum of features needed to support its domain.

• A DSL must have a specific **domain focus**. There is value of a limited language only if it has a clear focus on a small domain.

**1.1 Motivation**

DSLs are computer languages tailored to a specific application domain. Their use enables the expression of ideas and concepts within a domain by domain experts without extensive programming knowledge. DSLs often start simple, based on an initially limited understanding of the domain, but then grow more and more sophisticated over time.

The use of DSLs enables a better approach to model-driven development (MDD) than general-purpose modeling languages. A 2014 study found that the companies who successfully applied model-driven engineering largely did so by creating or using languages specifically developed for their domain, rather than using general-purpose languages such as UML (Hutchinson, Whittle, & Rouncefield, 2014). The creation of DSLs that are productive to use and address user needs is easier if languages and generators can be created in close collaboration with language users. The use of DSLs for MDD was found to improve productivity by 500-1000% as compared to using GPLs (Tolvanen & Kelly, 2016). There are several tools that allow for the definition and execution of DSLs, such as Eclipse Modeling Framework (EMF), Rational Software
Architect (RSA), MetaEdit+, and Generic Modeling Environment (GME), to name a few. However, before tooling needs are assessed, the model framework needs to be developed.

1.2 Challenges

The use of DSLs has increased manifold for problem-solving in specific domain areas as they allow for a wider variety of expressions within their domain. Modeling using DSLs has shown high increases in productivity after accounting for the time and cost expended in developing them, making them a suitable target for improvement in order to reap higher rewards (Thibault, Marlet, & Consel, 1999). DSL development is incredibly difficult as it requires knowledge of the domain and language development expertise, and usually few people have both. As a result, the creation of a DSL is often not considered and if so, a DSL rarely evolves into a full-fledged language that can be used within the domain it is intended for (Mernik, Heering, & Sloane, 2005).

DSLs support higher levels of abstractions than general-purpose modeling languages and are closer to the problem domain than they are to the implementation domain. Consequently, a DSL follows the domain abstractions and semantics, allowing modelers to perceive themselves as working directly with domain concepts. Furthermore, the rules of the domain can be included in the language as constraints, disallowing the specification of illegal or incorrect models (Mernik, Heering, & Sloane, 2005). Domain-specific modeling (DSM) is a manner of developing systems that uses DSLs to represent the various facets of a system, in terms of models (Čeh, Črepinšek, Kosar, & Mernik, 2011). However, DSM approaches mainly focus on structural aspects of DSLs. The behavioral semantics of models are not explicitly defined, which can lead to a semantic mismatch between design models and the modeling languages of analysis tools. This lack
of explicit behavioral semantics strongly hampers the development of formal analysis and simulation tools (Čeh, Črepinšek, Kosar, & Mernik, 2011).

The currently used approach for domain modeling involves the creation of an ontology which is then used to describe the domain model (Pereira, Fonseca, & Henriques, 2016). This ontology encapsulates all domain knowledge and can be cumbersome to create, requiring external sources of information and assistance from a domain expert (Čeh, Črepinšek, Kosar, & Mernik, 2011).

1.3 Research Questions

The following are the questions that drive the focus of the proposed research,

1.3.1 What is the use of standardized DSLs for scenario development in a given domain and what advantages does it offer?

DSLs are shown to increase productivity when used appropriately for a domain’s needs. The idea of a standardized DSL for a domain is studied from the perspective of the development of a DSL for aviation. A study of scenario generation for flight simulators shows that each research project utilizes its own method for this purpose which they develop from beginning to end. Many resources are being used for similar tasks which could be saved if there was a process to reuse some aspects of other projects. The idea of a common standardized language for defining aviation scenarios aims to reduce duplicate work and effort.
1.3.2 How can the process for DSL development be refined to leverage requirements elicited from domain users without additional domain expert assistance?

Currently, a lot of external research and collaboration between a modeling and simulation (M&S) expert and a domain expert is required to understand the specifics of domain structure and behavior to create a DSL. This part of the research investigates the use of existing documentation between developers and stakeholders to develop the DSL without external assistance. For this purpose, emphasis will be placed on the use of scenarios. Scenarios are used to describe the behavior of the system; first for eliciting requirements and later for testing. This requires the formal description of scenarios and the ability to understand the interaction between entities. Given that scenarios are already a part of the documentation of a DSL, considerable time and effort can be saved by using them as the basis for a domain model and adding entities as newer scenarios are described.

1.3.3 What value does the implementation of DSLs using only elicited requirements add in terms of a) time taken and b) technical knowledge required?

After investigating the possibility of using existing requirements documentation to develop a DSL model, the value of the contribution needs to be studied. This requires a case study detailing the benefits obtained from using such an approach compared to the current standard in terms of two important metrics: a) time taken and b) prior knowledge required. A thorough analysis of this approach can assist in reaching a conclusion about the practicality and applicability of the proposed solution.
1.3.4 How can a DSL be evaluated and the process for it formalized so that it can evolve over time?

Two major concerns with the development of a DSL are that: (a) there needs to be a mechanism to evaluate the effectiveness of the DSL for the domain, and (b) there needs to be a mechanism for adding additional elements to the DSL in the future, that is the framework for the evolution of the DSL. This question aims to provide such a framework so that the DSL can be an efficient and appropriate solution for the problems in the domain.

1.4 Contribution

This dissertation presents the design and development of a novel framework for domain modeling using scenario-based development techniques called the Domain-Specific Scenario (DoSS) Framework. The dissertation discusses operational scenarios that are already provided at the beginning of the modeling process and how they could be used to define a domain model. This approach proposes the use of scenarios in natural language, which are currently used in requirements engineering and testing, as the basis for developing a domain model iteratively.

The dissertation also discusses real-world applications of DoSS in the creation of an aviation-specific DSL called the Aviation Scenario Definition Language (ASDL). An Air Traffic Control training tool, the ATC Scenario Training Technology (ASTT) was used to understand the application of this DSL. Scenarios used in ASTT could be created using the ASTT metamodel, showing its applicability. The DoSS process has also been used to create an ontology-development framework using natural language scenarios.

Specifically, this dissertation involves the following four research tasks:
• Task 1: Aviation Scenario Definition Language. In this task, the design of an aviation-specific DSL is revisited from its original draft published in 2016. The process of development of ASDL is described and the advantages of a standardized DSL for a domain are discussed.

• Task 2: DoSS Design. In this task, the proposed scheme for the design of DSLs has been described and used to model a DSL for the aviation domain by incorporating additional scenarios into an existing aviation scenario language framework.

• Task 3: DoSS Evaluation. In this task, the same approach is followed by a user who is not an expert in the field in order to determine the practicality and usability of the proposed framework.

• Task 4: Formal Verification. In this task, the DSL obtained using the framework in Task 1 is verified formally with the use of statecharts in order to determine that it conforms to requirements.

1.5 Roadmap

The rest of this dissertation is organized as follows. Chapter 2 provides background information on DSLs and also presents a literature review of current DSL development techniques. Chapter 3 discusses ASDL and revisits the development process and extensions of ASDL as a possible standardized DSL for aviation scenario definition. Chapter 4 presents the proposed approach and a case study to demonstrate the framework. Chapter 5 presents a usability analysis and shows a comparison of the approach when performed by a user who is not an M&S or a domain expert. Chapter 6
describes the results obtained in the case study and discusses the value of the novel framework. Chapter 7 discusses some projects built on the research directions of this dissertation to show their applications and extensions. Finally, the concluding remarks and future research directions are reported in Chapter 8.
Chapter 2

Literature Review

This chapter first discusses DSLs and their development, and then it provides state-of-the-art research efforts in scenario-based development.

2.1 Domain-Specific Languages

DSLs are computer languages tailored to a specific application domain. They are essential assets of paradigms such as model-driven engineering (MDE) (Kelly & Tolvanen, 2008), and are frequently used in end-user development (Ko, et al., 2011). Modeling languages are often complex. For example, the Unified Modeling Language (UML) has metamodels with hundreds of associations, classes, and attributes (Rahman & Amyot, 2014).

DSLs capture the main features of a domain and enable succinct and natural system descriptions using a terminology close to the domain experts (Vaquero-Melchor, Palomares, Guerra, & Lara, 2017). The term ‘domain-specific language’ has also been used within the MDE community as a synonym for a ‘domain-specific modeling language’ (DSML). This is because the differences between DSLs and DSMLs are not unbridgeable and many similarities in the designs and implementations of DSLs and DSMLs can be identified (Kosar, Bohra, & Mernik, 2016).
A DSML is a DSL with a graphical concrete syntax for the primary purpose of diagrammatic modeling in a particular application domain (Strembeck & Zdun, 2009). A DSML focuses on providing modeling abstractions for the problem concerns in the application domain which are independent of a given software platform, rather than on issues of implementing the domain. Here, the term DSL has been used interchangeably with DSMLs.

2.1.1 Why are Domain-Specific Languages Used

The initial cost of DSL development is usually quite high compared to the equivalent cost of adapting a GPL for an application. DSLs are hard to build because the process requires domain knowledge and language development expertise, and few people have both (Mernik, Heering, & Sloane, 2005). However, in the long run, the slope of the curve for the aggregate software development curve should be considerably lower using a DSL, and thus at some point, the DSL approach should yield significant savings (Hudak, 1998).

DSLs are meant to close the gap between domain experts and computing solutions (Barišić, Amaral, Goulao, & Barroca, 2011). The shift of the developers’ focus to use constructs that are part of the real domain world, rather than general-purpose abstractions closer to the software specification world is said to bring important productivity gains when compared to using GPLs (Kelly & Tolvanen, 2000),

DSLs allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain (Van Deursen, Klint, & Visser, 2000). Consequently, domain experts themselves can understand, validate, modify, and often even develop DSL programs.
DSLs enhance productivity, reliability, maintainability (Van Deursen & Klint, 1998), and portability (Herndon & Berzins, 1988). They embody domain knowledge, and hence enable the conservation and reuse of this knowledge. DSL programs are concise, self-documenting to a large extent, and can be reused for different purposes (Ladd & Ramming, 1994).

One of the main goals while producing a DSL is to foster a more productive usage of the language by the users who will use it than GPLs or other existing alternatives. The interaction should favor an increase in the efficiency of people performing their duties without adding to organizational costs, inconveniences, dangers, and dissatisfaction for the user (Barišić, Amaral, Goulao, & Barroca, 2011).

2.1.2 How are DSLs Usually Created

There is no well-defined framework for creating DSLs currently. An oft-cited work in DSL development is an annotated bibliography (Van Deursen, Klint, & Visser, 2000) that surveyed 75 papers to determine a common process for DSL development and implementation. The findings of this paper are summarized first, followed by research in this field over the past two decades.

The development of a domain-specific language typically involves the following steps: Analysis, Implementation, and Use (Van Deursen, Klint, & Visser, 2000). Analysis involves the tasks of identifying the problem domain, gathering all relevant knowledge in this domain, clustering this knowledge in a list of semantic operations on them. Implementation is the process of actually constructing a library and designing a compiler that translates DSL programs to be executable. Use is the final step where applications
are written and compiled. A prerequisite to developing a DSL is mature domain knowledge (Van Deursen, Klint, & Visser, 2000).

A systematic mapping study on DSLs (do Nascimento, et al., 2012) found a large number of techniques, methods, and/or processes for creating new DSLs. Strembeck and Zdun (Strembeck & Zdun, 2009) found that the systematic development of DSLs is not fully tackled yet. Most existing approaches especially focus on technical facets of designing and implementing DSLs. In particular, they found that one of the main challenges for software engineers is to increase the participation of domain experts in DSL design activities, in order to avoid misinterpretations of the domain and/or designing a DSL detached from the needs of the domain experts (Strembeck & Zdun, 2009). They suggest that the core language model for any DSL captures all relevant domain abstractions and specifies the relations between these abstractions. In particular, these abstractions refer to elements of the DSL’s target domain. The DSL’s core language model thus formalizes domain-specific knowledge and must be validated by domain experts (Spinellis, 2001). The core language model can be defined using any suitable modeling language, such as the UML (The Object Management Group, 2007) and UML extensions. Depending on the type of DSL, the core language model can either be a metamodel or an ordinary model.

In most cases, DSL development is an explorative, iterative process (Strembeck & Zdun, 2009). Unfortunately, for different DSLs developed in different application contexts, the order in which these activities need to be performed, and the exact steps that must be executed to perform the activities can change significantly.
This dissertation focuses on the Domain Analysis step as this requires the most time and effort from the perspective of the DSL designer. The earliest suggested process was to start with a solid, simple design that should be tested by describing a wide variety of objects in the proposed language (Bentley, 1986). This baseline has been interpreted and implemented in various ways by DSL engineers.

In the analysis phase of DSL development, the problem domain is identified and domain knowledge is gathered (Mernik, Heering, & Sloane, 2005). The inputs at this stage are various sources of explicit or implicit domain knowledge, such as technical documents, the knowledge provided by domain experts, existing GPL code, and customer surveys. The domain is analyzed and then implemented as a set of domain-specific reusable components. An important task of domain analysis is commonality analysis which identifies useful abstractions that are common to all family members. Commonalities are the main source of reuse; thus there is an emphasis on finding common parts. Besides the commonalities, variabilities are also discovered during commonality analysis. Variabilities indicate potential sources of change over the lifetime of the family (Mernik, Heering, & Sloane, 2005).

The development of a domain-specific language typically begins with the domain analysis step (Cleaveland, 1988; Van Deursen, Klint, & Visser, 2000). The domain analysis method must provide specific representations to document the results of each of the domain analysis activities. These representations form a reference model for systems in the domain. The representations define the scope of the domain, describe the problems solved by software in the domain, and provide architectures that can implement solutions
In order to assign domain semantics to a language, it is necessary to specify the elements of the domain. This requires the creation of an ontology. Ontology is “the study of existence, of all kinds of entities that make up the world” (Sowa, 1999). A specific ontology is characterized as the set of elements of a particular domain and the relationships among them. Hence, mapping the elements of a modeling language to an ontology and vice versa provides domain semantics to that language.

A multi-method study (Sobernig, Hoisl, & Strembeck, 2016) aiming to extract reusable design decisions for DSLs found that a big challenge is to identify and to define the domain abstractions and their relationships in a manner which allows for: (1) a detailed specification independent from a concrete modeling language, and for (2) having the abstractions enter a UML-based language-model implementation in a seamless and/or guided way.

Sobernig et al. (Sobernig, Hoisl, & Strembeck, 2016) found that existing approaches towards systematizing DSML development put forth a development-process perspective, treating DSML development as a complex flow of exploratory and iterative development activities. Key activities are language-model definition, constraint specification, concrete-syntax design, and platform integration (Strembeck & Zdun, 2009).

Evermann and Wand (Evermann & Wand, 2005) say that the process for DSLs development begins with the analysis and description of the application domain. The
description of the application domain is formalized in a conceptual model, which can be used as the basis for subsequent software design and implementation.

In the context of software engineering, domain analysis is not primarily concerned with a faithful description of the application domain. Instead, it aims to develop a reusable and expandable software architecture or framework (Arango, 1989). It specifically takes into account software considerations such as reuse and configuration issues (Cohen & Northrop, 1998). This is the main difference between conceptual modeling and domain analysis.

2.2 Scenario-Based Development

The term “scenario” has a variety of published definitions from a vast array of domains. While the term itself has varying meanings, two definitions represent a scenario in the context of simulation development and usage. First, a scenario can be defined as a description of the hypothetical or real area, environment, means, objectives, and events during a specified time frame related to events of interest (GSD Product Development Group, SISO, 2014). Second, a scenario can be defined as a specification of conditions and situations to be represented by a simulation environment for its purpose (Durak, Topeć, Siegfried, & Oguztuzun, 2014). Both definitions agree that a scenario is a description of important events and conditions needed to represent a specific order of events which, in this application, occurs within a simulation environment. Therefore, when developing a simulation environment that utilizes these events and conditions, it is often a prerequisite to defining the scenarios that will be executed in the target simulation environment.
2.2.1 Scenario Definition

The Simulation Interoperability Standards Organization (SISO) released guidelines for the creation of simulation scenarios, detailed in (NATO Modelling and Simulation Group MSG-086, 2015). The guideline provides detailed information regarding the development of simulation scenarios, including an overview of available tools and processes.

According to (GSD Product Development Group, SISO, 2014), a scenario must be well-defined and must be complete, consistent, and comprehensible. Failing to achieve these aspects can lead to incomplete scenario definition and misunderstanding regarding the scope and applications of the simulation environment. As a result, the subsequently designed simulation environment is prone to error and may not reflect what the user originally intended. For a simulation environment that is designed to execute a specific set of scenarios, this misunderstanding can be disastrous.

Scenario development is an important part of all phases of the simulation environment development process. It not only defines a specification of a simulation run, but also provides an input for the design and evaluation of the simulation environment itself (Durak, Topcu, Siegfried, & Oguztuzun, 2014). Scenario development can be broken down into the creation of three scenario groups: (1) operational scenarios, (2) conceptual scenarios, and (3) executable scenarios. First, operational scenarios are provided by subject-matter experts (SMEs) in the early stages of development. These scenarios often provide a broad description of the desired events in textual form using natural language. For example, an operational scenario may describe what events occur during the simulation and in which order they should occur. Operational scenarios use domain-specific terminology to describe the events, which can then be identified by the M&S
expert and incorporated into the simulation ontology. Operational scenarios must be identified before development of the simulation environment, as virtually all simulation requirements are derived from these scenarios. Second, although operational scenarios provide the simulation requirements, they are too broad to provide the details necessary to derive a conceptual model and create a simulation environment. A conceptual scenario adds this detail and additional information to the operational scenarios. These scenarios are often created by an M&S expert in collaboration with an SME. Although similar to an operational scenario, conceptual scenarios should contain all information needed for the simulation environment. In addition, instead of a purely natural language approach, a conceptual scenario is used to create a conceptual model, which elaborates concepts into entities similar to UML classes. These entities contain properties and attributes which describe their roles and associations in the simulation environment. Finally, once the conceptual model is complete and the simulation environment is defined, executable scenarios can be made. An executable scenario is the specification of a specific situation providing all information necessary for preparation, initialization, and execution of a simulation environment (GSD Product Development Group, SISO, 2014; Durak, Topcu, Siegfried, & Oguztuzun, 2014; Fall & Fall, 2001). These scenarios are ideally specified in a way that allows them to be machine-readable and reusable. These types of scenarios can be seen in Figure 1.
Durak et al. (Durak, Topcu, Siegfried, & Oguztuzun, 2014) propose an MDE approach to the scenario development process using EMF. Following MDE principles, scenario development is viewed as the transformation of operational scenarios (defined using natural language) to conceptual scenarios (conforming to a metamodel) to executable scenarios (specified using a scenario specification language) and simulation environment design (following a specific formalism). As the development process occurs, the scenario models are refined and transformed. As a result, the proposed method requires conceptual scenarios to be based on a metamodel which is then transformed into executable scenarios for target simulation environments using a set of rules specific to the target simulation environment. Following this method of model-to-model or model-to-text transformations, a conceptual scenario can be transformed into multiple executable scenarios for multiple target simulation environments. This greatly promotes reuse and simplicity among the M&S community.

During the scenario development process, each scenario should at least specify three main components: (1) the initial state, (2) the course of events, and (3) the termination
conditions. As the scenario development process progresses, these three components should be further refined and expanded. The initial state describes the situation at the beginning of the scenario timeline and generally contains information such as date and time, surrounding conditions, and objects. The course of events describes any pre-planned events that occur at a specified time or in a specified order. These events can elicit a response, such as from a trainee using the system, or describe a change in entity association or state, such as an aircraft beginning a turn while en route. Events are often prompted by a trigger condition and each event injects changes to the simulation scenario state to achieve a desired effect or action. Any number of events can occur throughout the scenario timeline. The termination conditions describe the state of the simulation environment where the scenario can be defined as completed or terminated. This can occur via a specific event (such as a successful landing) or other means of measurement (such as achieving a predefined elapsed time) (GSD Product Development Group, SISO, 2014).

Scenario-based development and the scenario development process are complex and time-consuming. However, a variety of tools and standards exist to help alleviate these challenges. For operational scenarios, the Distributed Simulation Engineering and Execution Process (DSEEP) describes a generic 7-step process for developing and executing a simulation environment. DSEEP was used by Durak et al. (Durak, Topcu, Siegfried, & Oguztuzun, 2014) when implementing the MDE approach to scenario development. The Coalition-Battle Management Language (C-BML) strives to provide a standard for specifying the course of events within a scenario. For conceptual scenarios, the Base Object Model (BOM) defines a standard for defining and reusing components of
models, simulations, and federations. BOM can be used as a base for a simulation conceptual model and in the design of scenarios for interoperable simulations. For executable scenarios, the Military Scenario Definition Language (MSDL) is the most well-known standard for specifying executable scenarios in the military domain (GSD Product Development Group, SISO, 2014).

2.2.2 Scenarios in DSL Development and Implementation

There has been some research on the idea of using scenarios in DSL development and implementation. However, there is no defined process for this purpose. Some of the key conclusions from relevant projects have been summarized in this section.

Scenarios are considered useful in the software engineering process because they divide the whole business requirement into smaller chunks of scenarios and hence the processes and rules are divided as well. This makes the model more flexible and owns strong scalability because a new requirement could be a new scenario so existing documents do not need to be changed. This also leads to a division of development tasks for different departments at the same time, which could help save some time from requirement analyzing (Xi, et al., 2019).

El-Kechai and Choquet (2007) say that scenarios carry a specific but implicit representation model. This should be made explicit and can be leveraged for enabling the reuse and the sharing of the scenarios expressed among this model. They proposed an approach for reusing pedagogical scenarios for teaching expertise capitalization by formalizing a UML model into an XML schema, but the approach and result were specific to the education domain.
Ogata and Matsuura (2008) proposed a method for automatically generating a prototype system for requirements analysis. They proposed a stepwise development of the requirements analysis model and the automatic generation of the prototype system, which would include appropriate scenario-based concrete examples to help the developers understand the relationship between them. The initial prototype system indicates the service flow of the use case. The second prototype system is automatically generated by combining the activity diagram and the class diagrams and indicates the service flow, the related data type, and the structure of the use case. The third prototype system, which is automatically generated by combining the activity diagram, class diagram, and object diagrams, indicates the service flow and the related concrete data example of the use case. The developer then generates an activity diagram to specify the execution order of use cases for each actor. The whole prototype system is then automatically generated by these four diagrams and indicates a scenario of workflow for each role of user. The customer confirms the requirements analysis model as meeting the requirements by using the prototype system.

Textual scenario descriptions are used throughout the process of software development, with different abstraction levels and use cases depending on the particular phase of development (Bock, Sippl, Heinzz, Lauerz, & German, 2019). Textual information provides better readability, higher writing efficiency, and can save space, although different graphical approaches exist as alternatives for scenario modeling (Grönninger, Krahn, Rumpe, Schindler, & Völkel, 2014).

Bagschik et al. (2018) have worked on scenario development where an ontology is used to describe scenarios formally. After the ontology for the given use case is modeled, the
process for the creation of scenarios uses the domain knowledge to create valid traffic
scenes. The scenarios are then used to analyze the system properties. This approach can
be used for verifying and validating an autonomous driving simulation system through
these scenarios, which is the next step of this project.

Two major projects relating to the use of domain-specific languages and scenario-based
development have been published in the last two years. Their main processes,
conclusions, and relevance to the DoSS effort have been described below.

The stiEF (scenario-accompanied, text-based, iterative Evaluation of automated driving
Functions) methodology (Bock, Sippl, Heinzz, Lauerz, & German, 2019) was first
published in 2019 and is still in its early development stages. The paper discusses the
advantages of textual DSLs for scenario-based development. Some recent research
activities have proposed systematic methods of scenario creation, either by using an
ontology and knowledge-based method (Bagschik, Menzel, & Maurer, 2018) or by using
a holistic workflow. While these methods allow the identification and manual creation of
scenarios, a proper workflow to iteratively refine textual scenario descriptions in natural
language and to ensure their completeness is still missing (Bock, Sippl, Heinzz, Lauerz,
& German, 2019).

One of the advantages Bock et al. (2019) have discussed is that every domain expert
familiar with the domain’s terminology can create textual descriptions, whereas modeling
in a specific format such as SysML requires specific modeling knowledge. This means
text-based languages have an advantage over graphical and symbolic formats such that
tool or technological expertise is not required. However, domain expertise is still needed
in this case.
The stiEF methodology follows the following steps (Bock, Sippl, Heinzz, Lauerz, & German, 2019): (1) the information from all scenarios is imported as textual data, (2) the scenario data are parsed with regards to the meaning and context, and (3) the information is merged into the existing scenario catalog.

While this methodology also relates to the use of DSLs and scenario-based development and testing, it differs from the research in this dissertation because it uses an already-defined textual DSL to parse natural language scenarios. These scenarios are then intended to refine and extend the DSL.

The Association for Standardization of Automation and Measuring Systems (ASAM) OpenSCENARIO v1.0 standard (ASAM, 2020) defines a data model and a derived file format for the description of scenarios used in driving and traffic simulators, as well as in automotive virtual development, testing and validation. It is defined using UML.

A concept paper for OpenSCENARIO v2.0 published in 2020 (ASAM, 2020) proposed the second version to be founded on the concept of a DSL that should support all levels of scenario description, from the very abstract to the very concrete in a suitable way. The OpenSCENARIO 2.0 concepts aim to take proven features and capabilities of OpenSCENARIO 1.0 and place them in a more general and expressive language framework, to serve as the foundation for both incremental improvements and more revolutionary enhancements.

The proposed DSL will support the creation of more complex scenarios through flexible composition and parametrization of component scenarios. This is expected to enable testing and verification of new, complex hardware and software systems, and their
interaction with the complex environment of driving (ASAM, 2020). The resulting DSL is expected to have a modular structure in order to support reuse and concurrent engineering. However, there is no published or specified framework for the creation of this DSL.

2.3 DSL Evaluation and Verification

No DSL is ever finished (Voelter, et al., 2013), there is always room for improvement (Bock, Sippl, Heinzz, Lauerz, & German, 2019). A survey on the assessment of DSLs points out the absence of a systematic assessment of the languages (Kahraman & Bilgen, 2015). The growing number and complexity of DSLs raise the necessity of a systematic approach for assessment of DSLs (Kelly & Tolvanen, 2008; Strembeck & Zdun, 2009). Most software evaluation and inspection models are independent of software development procedures and artifacts (Taba & Ow, 2012).

Despite the importance of having effective DSLs, the quality of a DSL is an in-progress concept. The existing research evaluates how well DSLs perform in use and lists desirable or undesirable properties that can be found in good or bad DSLs (Kahraman & Bilgen, 2015).

Mohagheghi et al. (Mohagheghi, Dehlen, & Neple, 2009) performed a literature review on approaches to model quality in model-based software development. However, their scope was limited to UML models as UML is considered to be the most widely used modeling language. They concluded that the comprehensibility of models can be improved by adding semantics to an informal language that facilitates analysis and generation. Using a language close to the domain improves comprehensibility by humans, especially by non-technical domain experts. The solution domain gets closer to
the problem domain which also improves the maintainability of models. This suggests that model-based software development can greatly benefit from the use of DSLs for software solutions.

A systematic literature review on the usability of DSLs (Rodrigues, Campos, & Zorzo, 2017) suggested that there is a lack of formal structure in the usability evaluation of DSLs. Barisic et al. (2014) introduced a methodology that considers usage quality criteria from the beginning of DSL language development. Their study considers usability criteria during the whole development of the DSL and also during the modeling of a system as an application of the DSL. Their main conclusions were that it is necessary to have a clear definition of the quality criteria that will be used to evaluate the DSL which they found to be missing.

Albuquerque et al. (2015) suggested the use of quantitative and qualitative methods for DSL evaluation. They presented an evaluation method called Cognitive Dimensions (CD) containing 14 dimensions: viscosity, visibility, compromise, hidden dependencies, expressiveness role, error tendency, abstraction, secondary browsing, mapping proximity, consistency, diffusion, hard mental operations, provisional, and progressive evaluation.

Barisic et al. (2014) found a big error rate when inexperienced users used the proposed language. One of the conclusions from the authors was that it could have been caused by the lack of feedback that the tool provides for users. Therefore, it is important to provide users who are not domain experts with feedback and necessary information to correct any errors they might encounter.
A proposed approach to solve the problem of ad hoc testing lies in automated testing and the use of specialized test languages and formal specifications. Ryser and Glinz (1999) proposed the use of scenarios, not solely for requirements elicitation and specification, but specifically for system testing. The proposed method is called a method for SCENARIO-Based Validation and Test of Software (SCENT). The SCENT method (Ryser & Glinz, 1999; Ryser & Glinz, Using dependency charts to improve scenario-based testing, 2000; Ryser & Glinz, SCENT: A method employing scenarios to systematically derive test cases for system test, 2000) explicitly mentions that the idea of the usage of scenarios for testing is previously established (Jacobson, 1993; Jacobson, 1994), but a process had not been formally defined until the SCENT method was published.

The SCENT method aims to use scenarios not only to elicit and document requirements but also to describe the functionality and specify the behavior of a system and to validate the system under development while it is being developed. SCENT intends to uncover ambiguities, contradictions, omissions, impreciseness, and vagueness in natural language descriptions by formalizing narrative scenarios with statecharts (Harel, 1987). The final step is to systematically derive test cases for system tests by traversing paths in the statecharts, choosing a testing strategy as appropriate, and documenting the test cases. A big advantage of this method is that contrary to other approaches (Rachida, Dssouli, & Vaucher, 1996), SCENT does not restrict and formalize natural language to capture requirements (Ryser & Glinz, Using dependency charts to improve scenario-based testing, 2000). Instead, it relies on the developer to synthesize statecharts from the information in natural language scenarios. The formalization step performed by the
developers is the transformation of structured natural language scenarios into a statechart representation.

Ryser and Glinz (SCENT: A method employing scenarios to systematically derive test cases for system test, 2000) chose statecharts for verification as these are an extension to well-known state diagramming techniques. They help reduce the explosion in the number of states that occurs in state automatons having many parallel processes. Furthermore, they introduce the ability to decompose complex models. Statecharts may be defined as state diagrams enhanced with the capacity to decompose complex models hierarchically and to model parallel processes (Harel, 1987). Creating statecharts out of scenarios is modeling, and as it is a creative and innovative activity, which is why it has been considered harder to formalize. However, it is an inherent part of the modeling process and helps translate natural language user stories and scenarios to a visual format of system changes. A single step in a scenario usually translates into a state or a transition in a statechart. As the steps are mapped to either states or transitions, missing states and transitions will emerge and need to be added. Superfluous states (and transitions, if any) need to be deleted or merged with needed states (or transitions).

The visual format makes it obvious when mistakes are made. It is easier to see if all the necessary states are specified and if all the states in a statechart are connected. Normally the first statecharts created of a system will have to be corrected, improved, and revised. At first, the developer verifies the statecharts against the specification, that is the scenarios. Then the statecharts have to be reviewed and validated by the customer and user.
The verification of statecharts against the scenarios they are derived from is a non-trivial problem. There is no given/fixed mapping between the two. Correspondingly, statecharts cannot be verified automatically and need to be reviewed manually against requirements. Even though statecharts may not be verified automatically, yet it is valuable to create statecharts, because they may be checked internally for consistency, internal completeness, and the fact that invariants and restrictions are met.

The SCENT method was novel for its use of artifacts of the early phases of the development process in testing again and because it handily integrates with existing development methods. The method was applied in practice (Itschner, Pommerell, & Rutishauser, 1998; Ryser & Glinz, 1999; Ryser & Glinz, Using dependency charts to improve scenario-based testing, 2000) and was considered to have been met its main goal, namely to supply test developers with a practical and systematical way to derive test cases from natural-language scenarios.

The use of scenarios was perceived by the developers as helpful and valuable in modeling user interaction with the system. The formalization process however posed some problems, as the mapping of actions in natural language scenarios to states or transitions is not definite and clear-cut. A narrative scenario transformed into a statechart by one developer may differ significantly from a statechart developed from the same scenario by another developer.
Chapter 3

Aviation Scenario Definition Language

This chapter discusses the creation of an aviation-specific DSL called ASDL (Jafer, Chhaya, Durak, & Gerlach, 2016). It describes the use of defined DSL methodologies listed in the previous section to present a simpler, standardized method for aviation scenario generation. Based on DSL design methodologies, ASDL provides a well-structured definition language to define aircraft departure, en route, re-route, and landing scenarios.

3.1 Background

Two different concepts need to be introduced as the background of the approach for creating ASDL. The first one is DSLs which are computer programming languages with a limited expressiveness focused on a particular domain (Fowler, 2010). They are developed based on the concepts, terms, and relations among them in that particular domain, which is called a domain model. Ontologies are commonly used for domain modeling (Durak, OĞUZTÜZÜN, & Ider, 2009). A major advantage of DSLs is that they allow non-developers and users who are not well-acquainted with the domain to understand the overall design.

The second concept is that of scenario generation. As noted by Jafer et al. (2016), several researchers are working independently on scenario generation, who tend to use various
simulators and associated tools and create customized data processing programs in order to modify an existing data set into the desired scenario. This leads to the drawback that each scenario generator is currently coupled with the application and simulator it is being used for. Also, it appears that each of these research projects utilizes its own method for this purpose which they develop from beginning to end (Signor, et al., 2004). Many resources are being used for similar tasks which could be saved if there was a process to reuse some aspects of other projects. This leads to the conclusion that if there was a common standardized language for defining aviation scenarios, a lot of duplicate work and efforts could be avoided. Each related project (for example air traffic scenarios, runway collision avoidance scenarios, and full-flight simulation scenarios) could use relevant information (airport, runway, flight plan objects) from this language in order to build their specific scenarios which could then be ported to and read by any simulator they choose. This is the main purpose of creating ASDL, which could benefit all aspects of the aviation community for modeling and simulating the various features of flight. ASDL has been briefly introduced by Jafer et al. (2016); in this section, a more detailed view of the internal mechanism of ASDL has been presented.

3.2 Approach Methodology: Ontology

In philosophy, the term ontology has the meaning of a systematic explanation of existence. Gruber defined it for computer science as: “Ontology is an explicit specification of a shared conceptualization” (Gruber, 1995). 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 (Noy & McGuinness, 2001). The merits of ontologies are:
a) common vocabulary, b) explication of what has often been left implicit, c) systematization of knowledge, d) standardization, and e) meta-model functionality (Mizoguchi, 2001).

In order to capture the concepts and their relations in a flight simulation, it is essential to have a definition reference list that highlights all key terminology as well as procedures and operations that are communicated between the pilot and Air Traffic Control (ATC). The United States’ Federal Aviation Administration (FAA) and Single European Sky ATM Research (SESAR) programs provide inclusive glossaries that specify key terminology and concept of operations (Federal Aviation Administration, 2012; SESAR). In addition, the National Aeronautics and Space Administration (NASA) has developed a data integration system using ontologies for Air Traffic Management (ATM) data (Keller, 2016). This NASA ATM Ontology includes terminology and concepts to interconnect data from several different aviation realms, including flight, traffic management, aeronautical information, weather, and carrier operations. For this purpose, it includes details of flights, aircraft, airlines, airports, and all of the United States’ National Airspace System (NAS) facilities for ATM. This is a highly comprehensive ontology prepared in association with the FAA but has a strong focus on ATM. Extensive technical documentation has also been published to supplement this ontology which could be used to recreate the effort, but it was not available at the time ASDL was conceptualized (Keller, 2017).

With this motivation, an ontology was constructed (Jafer, Chhaya, Durak, & Gerlach, 2016) using the Web Ontology Language format (OWL), which was further utilized to develop the metamodel the ASDL. OWL is a commonly employed ontology language
that enables describing a domain in terms of classes, properties, and individuals and may include rich descriptions of the characteristics of those objects (Bechhofer, 2009). The current version of ASDL ontology has four base classes: Air_Traffic_Control, Aircraft, Airport, and Weather, all of which have further sub-classes and attributes. Full details of current ASDL ontology can be accessed online¹.

Simulation platforms may have varying levels of details for model behavior. For this reason, a basic ontology has been developed for conceptual models here which includes the bare minimum details required to describe a flight from takeoff to landing. Any new information that could be provided in a higher-fidelity simulation platform should result in an additional element being added to the ontology, so it can continue to be built to include all details of the aviation domain.

### 3.3 Model Design and Behavior

This section documents the design of a model created using ASDL, starting with the design process and then explaining the metamodel created and its properties.

#### 3.3.1 Base Object Model

Scenarios are generally distinguished into three phases of refinement: operational scenarios, conceptual scenarios, and executable scenarios (GSD Product Development Group, 2014). Operational scenarios are provided by the users or subject-matter experts and are commonly in simple textual or graphical forms. They specify what needs to be represented in a simulation environment in order to meet the user’s objectives. These operational scenarios need to be refined and augmented with additional information in

¹ Accessible online at https://github.com/ASDL-prj/Ontology
order to derive a conceptual model and design a simulation environment. This refinement is usually done by M&S experts and results in conceptual scenarios. An executable scenario is the specification of a particular situation that provides all the details necessary for the preparation, initialization, and execution of a simulation environment. The transformation from conceptual scenarios to executable scenarios is done primarily by the operators of the member applications of the simulation environment. Ideally, the resulting executable scenarios should be specified in a way that they are directly accessible by the member applications as a file.

The development of simulation models is a time- and resource-intensive task. Consequently, there has been a need for “modeling and simulation environments to be constructed from affordable, reusable components interoperating through an open system architecture” (Gustavson & Chase, 2004). SISO focuses on methods that promote the reuse of simulation components, agile and efficient development and maintenance of models, as well as the integration of models into operational systems. To this end, SISO developed the concept of BOM and released a specification and user guide for it in 2006. The objective of BOM is to provide a component framework for facilitating interoperability, reuse, and composability (SISO Base Object Model Product Development Group, 2006). It is based on the idea that different pieces of models and simulations can be extracted and reused as building-blocks or components in the process of modeling. BOM uses patterns to capture the interplay within a simulation. These reusable patterns of interplay are sequences of events between simulation elements. A BOM consists of a group of interrelated elements divided into four categories: Model Identification, Conceptual Model Definition, Model Mapping, and Object Model
Definition. Each category has multiple components that define the properties of the BOM. The major components have been illustrated in Figure 2.

![Figure 2: BOM composition](SISO Base Object Model Product Development Group, 2006)

Following Siegfried et al. (2013), BOM was utilized as a basis for defining conceptual scenarios. The Conceptual Model Definition identified above includes four components. Figure 3 shows the top-level view of the conceptual scenario in the BOM metamodel used to create ASDL. These four main components are defined below (SISO Base Object Model Product Development Group, 2006).
1. Pattern of Interplay: It identifies sequences of pattern actions (including variations and exceptions) which take place between sending and receiving conceptual entities, and are necessary for fulfilling a pattern of interplay. A Pattern of Interplay contains one to many PatternActions, each consisting of a name and a sequence. Variations and Exceptions to these PatternActions are also defined in the BOM metamodel.

2. State Machine: It identifies the different behavior states that are expected to be exhibited by the conceptual entities which are required to represent aspects of the conceptual model. Each state has an exit condition that leads to the next state. This exit condition is defined in the form of a pattern action from the pattern of interplay.

3. Entity Type: It identifies the types of conceptual entities required to represent aspects of the conceptual model. All entity types have their own characteristics, which is a separate model element.

Figure 3: Top-level view of ConceptualScenario in BOM metamodel
(Durak, Topcu, Siegfried, & Oguztuzun, 2014)
4. Event Type: It identifies the conceptual events, either directed or undirected, which are required to represent the pattern actions associated with a pattern of interplay. Each event requires a source and may or may not have a target. The execution of a directed event requires a trigger condition.

The structure of BOM allows a number of communicating components to be defined. The pattern of interplay describes the sequence of events including the sending and receiving entities as well as the event messages and triggers. In some cases, an action might cause a change in the state of the object or entity. As all of these can be explicitly described using BOM, it enables a complex set of scenarios to be modeled. For ASDL, this is particularly useful to model multiple aircraft, whether landing, taking off, or cruising, along with the ground interactions involved for each flight. The integration of the BOM metamodel with ASDL entities has been described later in this chapter.

3.3.2 Design Process

A typical scenario development process follows the steps shown in Figure 4. According to this chart, generating a conceptual scenario requires the definition of the simulation environment objectives and conceptual analysis of the same.
The design and implementation of the ASDL metamodel were based on the workflow provided by Durak et al. (2014). To reiterate, the steps followed were: definition of classes required to accurately represent the model, determination of the attributes used to describe the classes, the definition of the structure and relationships between the classes, creation of a metamodel based on the entities identified, integration of this metamodel of aviation entities into the BOM framework constructed by Durak et al. (2014), generation of Java code for the model, creation of a runtime instance, and its use to define and edit an aviation scenario.
The framework was developed in two main stages, as follows:

- **Stage 1**: 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.

- **Stage 2**: The aviation metamodel was integrated with BOM metamodel in order to define scenarios with specific aviation-related properties.

The design of the metamodel will be described in the remainder of the section.

### 3.3.3 ASDL Metamodel

The overall view of the aviation-specific metamodel of ASDL and all the conceptual entities have been described in this section.

The Scenario class is the base class that contains all the other classes in the metamodel. An instance created of this class can define the properties of all other classes and their attributes. It allows users to define three different kinds of scenarios: departure, rerouting, and landing. It also includes pilots, airports, runways, control towers, flight properties, weather patterns, and aircraft, which collectively define all the required scenarios.

The aircraft class is connected to several other classes and contains various attributes that define its properties. The basic structure can be seen in Figure 5. An aircraft has one attribute – its call sign – and has various associations with other classes. It is connected to two airports – its port of origin and destination. The runway that the aircraft is meant to land on is also specified in the model. At any given point, the properties of the flight and weather are also associated with the aircraft class. All aircraft must have a pilot and are associated with the ATC tower which is following them at the given time. In addition to
these properties, an aircraft also has a state at all times, which describes the behavior of
the aircraft at that time.

![Figure 5: Properties of the Aircraft class](image)

All the entities attached to the aircraft need to be defined individually in order to include
all relevant information about them. The airport class contains one attribute – a string
identifier for the airport code. It is also connected to various (zero or more) aircraft that
are departing from and arriving at it. Each airport has an air traffic control tower and one
or more runways. A runway has three properties – an identifier that includes its heading,
the name of the airport that operates it, and the length of the runway. The ATC class
includes the controller who is acting on behalf of the tower, the airport that it is attached
to and the zero to many aircraft that it is currently monitoring and instructing. The pilot
class consists of the name of the pilot and the aircraft they are associated with. This class
would invoke the method to make changes to the state of the aircraft in response to the
instructions from ATC. The FlightProperties class shows the instantaneous properties of
the flight. It includes the location as well as the motion and speed characteristics. The
attributes included in this class are the ones that were defined under the flight properties
section of the ontology. The weather class shows the instantaneous weather conditions
around the aircraft. It includes the location in addition to various properties that describe the weather, such as temperature, windShear, crossWindComponent, dewpoint, visibility, and skyCondition. The attributes included in this class are the ones that were defined under the weather section of the ontology.

3.3.4 Integrated Metamodel

The final metamodel was created by integrating ASDL and BOM metamodels. This section describes the main classes that were added to the overall metamodel. The two metamodels were connected by including the three types of aviation scenarios (DepartureScenario, LandingScenario, and ReroutingScenario) under a single base class called ConceptualScenario. In addition, all the aviation entities were grouped under a single ConceptualEntity class as discussed below. The resulting metamodel was too large to be depicted appropriately in a single diagram and has been broken down (Figure 6) for ease of understanding.

The base class in the integrated model is ConceptualScenario. An instance created of this class can define the properties of all other classes and their attributes. In addition to the three main types of aviation scenarios, this class also includes all other classes, such as other conceptual entities, patterns of interplay among various entities, a state machine that defines the states of an entity, events which cause state changes as well as a class listing all the basic attributes of the scenario. The ScenarioIdentification class lists a number of attributes that explain the scenario in terms of its name, version, description, purpose along with a point of contact and their details. This class is meant for informational purposes. Both of these classes can be seen in Figure 6.
Among other important classes, the ConceptualEntity class has a name and characteristics of the entity. It defines all the domain entities that interact with each other in order to perform actions within the scenario. All the aviation scenario classes defined in the previous section count as conceptual entities and have been added as children of this class. The PatternOfInterplay class provides a mechanism for identifying sequences of pattern actions (including variations and exceptions) which are necessary for fulfilling a pattern of interplay, as discussed within the section on BOM. A PatternAction is a single step in a pattern of interplay that may result in a state change of a conceptual entity. It has a name, a position in the sequence of the pattern of interplay, a conceptual entity as a sender, and another as a receiver, an event that represents the pattern action as well as related variations and exceptions. This class identifies a pattern action to be carried out in
order to successfully accomplish the named pattern of interplay that it is a part of. The StateMachine class and Events class have both been discussed within the BOM section.

3.4 Standardization Process

With the completion of the ASDL extensions, full flight operation scenarios can now be specified formally. However, ASDL must be standardized for it to be of use to the M&S community. A well-known problem in the aviation simulation community is the complexity of simulators and variety in simulator designs. This commonly results in individual implementations of scenarios that work only on a specific simulator system. Standardization of ASDL creates a more accessible means of defining aviation scenarios which are reusable, easier to understand and reduce error through reduced complexity. By following the SISO standardization guidelines and procedure, ASDL can become a full-fledged standard providing a general, stable, and supported means of defining scenarios for use in aviation in ways that are not limited to a specific simulator design.

SISO is an international organization dedicated to the promotion of interoperability and reuse in the M&S community (Simulation Interoperability Standards Organization - SISO, 2017). SISO publishes a variety of standards that strive to meet this goal in a variety of simulation domains, including analysis, research and development, testing and evaluation, and training (Simulation Interoperability Standards Organization - SISO, 2017). These standards provide a common base for creating reusable simulation components. However, becoming a SISO standard is not an easy task. SISO standards need to be stable, well understood, technically competent, have multiple independent interoperable implementations, be well received by the community, and be recognizably
useful to the M&S community (Simulation Interoperability Standards Organization - SISO, 2017).

There is a six-step process to becoming a SISO standard: (1) Activity Approval, (2) Product Development, (3) Forming Balloting Group and Product Balloting, (4) Product Approval, (5) Interpretation, Distribution and Configuration Management, and (6) Periodic Review. After being nominated to become a SISO standard, the product must apply for formal SISO approval. The proposal may be based on work previously done by an external organization. If the proposal is approved, the product begins development, following SISO standards, by a Product Development Group (PDG), which is assigned to the project. The PDG then presents the status of the product for approval to begin balloting to become a full-fledged standard. If more work on the product is needed, it goes back to the product development step. Otherwise, it moves into product approval. If the product demonstrates adherence to SISO principles for inclusion as a SISO standard, it is accepted by a committee. Once approved, a Product Support Group (PSG) takes the responsibility of the product and handles the distribution and configuration management for the product, and provides support to the community for the product. Finally, the PSG will periodically review each product to ensure they are still relevant to the community and they continue to meet SISO requirements (Simulation Interoperability Standards Organization - SISO, 2017).

In 2008, the MSDL standard was accepted and published as a SISO standard. Reaffirmed in 2015, this standard defined an XML-based language for defining military scenarios. The intent of MSDL was to create a general and reusable method of creating and verifying military scenarios, improve consistency among scenarios, and create reuse of
scenarios between different simulators through a standard scenario language format (Simulation Interoperability Standards Organization: Standard for Military Scenario Definition Language (MSDL), 2008).

ASDL was created with MSDL as a source of inspiration, as both languages share a similar domain of scenario generation and application. However, ASDL addresses scenario generation for the aviation domain, which is not addressed or supported through MSDL. Similarly to MSDL, in this research project ASDL is defined using an XML schema that allows for format standardization and content verification. This definition of ASDL will be used towards applying for SISO standardization.

3.5 Advantages

Once all properties have been defined, the model can be validated to see that all the required properties have values that are within the defined bounds. This only checks that all objects have values for mandatory properties and that these values conform to the standards defined in the metamodel. For example, an Aircraft without an attached Origin or Destination would not pass this automated validation check. This check for completeness is placed within the ASDL metamodel. An additional check can be performed by including input validation in the interface used directly by the end-user to enter data.

Current scenario development practices in flight simulators suggest a need for a common scenario specification language and process to improve consistency and enable reuse of scenarios. This chapter provides a detailed overview of a newly published DSL called ASDL and its implementation. ASDL was designed using the BOM framework to create a scenario generation metamodel for aviation applications.
The initial framework presented in this chapter is highly extensible. The results of this work have been recently used in developing a graphical modeling environment and to automatically transform scenario models into executable scenario scripts. The overall goal is the recognition of ASDL by the aviation community as a standardized scenario definition language.

The American Institute of Aeronautics and Astronautics (AIAA) Modeling and Simulation Technical Committee (MSTC) recently launched a working group towards the development of a standard simulation scenario definition language for aviation which addresses the lack of standardized practices and common format that lead to degraded interoperability and shareability (Torens, et al., 2018). The AIAA MSTC workgroup on simulation scenario development aims at a well-formed standard language for specifying simulation scenarios. (Durak, et al., 2018). This effort for the creation of ASDL and the idea of a standardized DSL for aviation has received 20 independent citations.
Chapter 4

Proposed Approach

This chapter presents the approach methodology and initial research results of a scenario-based framework for modeling DSLs. It discusses the use of a scenario-based approach to domain-specific modeling. In order to use natural-language scenarios to drive development, these scenarios need to be suitably broken down to extract information about the functionality of the language and domain model. In this case, each sentence of the scenario narrative will be studied to determine the parts of speech present in it, specifically, nouns, verbs, and prepositions. This use of the same scenario text to define both the static model and the dynamic behavior will reduce the number of artifacts to be produced in order to have a functional domain model design, and will ensure that both aspects of the domain are being developed from the same source material.

4.1 Problem Statement

The current approach for the creation of a DSL relies on the creation of an ontology, which requires external domain knowledge before the domain modeling process can be started. This dissertation proposes a scenario-based approach to modeling for domain-specific languages, such that the same scenarios used for requirements and testing are used to design the domain model.
4.2 Approach Methodology: Domain-Specific Scenario Framework

A new framework for developing domain-specific languages is presented in this section and is called the Domain-Specific Scenario (DoSS) Framework. It requires the use of scenarios that the language intends to model in the creation of the domain model as shown in Figure 7.

![Figure 7: Process for creating a domain model](image)

The steps to be followed using the DoSS framework can be seen in Figure 8.

![Figure 8: Steps outlining the DoSS framework](image)
DoSS proposes the following steps in order to arrive at a domain model from the scenario specification:

1. **Identify all entities and their values in the scenario.** Routinely, this includes all the nouns used in the definition of the scenario.

2. **Determine the interactions between these entities.** Interactions are usually defined by the use of a verb or a preposition connecting two nouns, either entities or their specific values in the scenario.

3. **Define elements based on interactions.** Ideally, any entity directly interacting with more than one other entity should be a parent entity or a model object. An entity that is connected to just one other entity can be considered either a model object or an object’s attribute. This definition may change as new scenarios are considered during the design process.

4. **Combine all elements and interactions.** Each entity’s interaction with all other entities is drawn to create the model centering on that particular entity. All these entity diagrams are then combined to obtain the domain model based on the given scenario. A sample of scenarios that make up the requirement model should contain all the necessary information about the domain model, and combining these domain models for all scenarios should provide the model for the DSL.

In order to use the natural-language scenario to drive development, it needs to be suitably broken down to extract information about the functionality of the language. In this case, each sentence will be studied to determine the parts of speech present in it, specifically,
nouns, verbs, and prepositions. Each fragment of the scenario that contains such information will be noted down and used to construct tables in two forms:

1. A table for domain model development, which will focus on entities and their interactions, which will ultimately lead to the generation of a domain model diagram for the DSL; and

2. A table for domain behavior definition, which will focus on the sequence of events that lead to changes in entities, which will ultimately lead to the generation of a statechart showing the domain behavior.

This use of the same scenario text to define both the static model and the dynamic behavior will reduce the number of artifacts to be produced in order to have a functional design, and will ensure that both aspects of the domain are being developed from the same source material.

4.3 Preliminary Discussion

This section discusses the preliminary use of a scenario-based approach to obtain a domain model of ASDL. First, the work done using the ontology-based development framework discussed in Chapter 3 is summarized, and then DoSS is used to obtain another domain model for ASDL. The two approaches are then compared to determine the differences in the final model, the time taken, and the number of scenarios required in order to sufficiently model the domain.

4.3.1 Ontology-Based ASDL

The ontology-based approach to ASDL has been documented by Jafer et al. (2018) and Jafer, Chhaya, & Durak (2017). First, an ontology was created containing all keywords
related to a flight scenario and the relationships between them. This ontology and the relationships were then used to create a domain model in Eclipse. Finally, the scenarios were modeled using the Eclipse domain model as an underlying metamodel.

The ASDL ontology has currently been extended to contain 149 elements in three stages: (1) the addition of landing terms and modeling of a landing scenario as a case study, (2) the addition of departure terms as part of an independent study into language extension, and (3) the addition of en route and reroute terms in order to complete the definition of a flight scenario from departure gate to landing gate.

This ontology has been used to create a domain model for ASDL in EMF. For this preliminary discussion, only the first stage of the ontology and domain model has been considered (landing case study). Some of the important terms in the landing ontology and domain model are shown in Figure 9 and Figure 10 respectively.
This section shows the generation of a domain model for aircraft landing using the DoSS framework. The process is described for a single scenario, and then other scenarios are used to add to the domain model produced at the end of the example.
4.3.2.1 Scenario 1

A sample scenario that can be defined using this model is described here. This is the first normal landing scenario, and will be referred to as NL01 henceforth:

*Aircraft ER-1357 has departed from the Daytona Beach International Airport (KDAB) via runway 7L. ER-1357 will be landing on runway 29 at the Gainesville Regional Airport (KGNV). Stable weather conditions along the route are reported as calm winds, dew point: 52, sky condition: scattered clouds at 5500 feet, temperature: 65, visibility: 10. Once within 30 miles of KGNV, Jacksonville approach control contacts ER-1357 and the landing phase begins. The pilot of ER-1357 will need to contact ATC for descent, approach, and landing clearances and will land normally without any complications.*

It can be seen by looking at Figure 10 that all elements that are described in the scenario have a corresponding entity in the domain model and hence can be modeled directly. This scenario will be used to illustrate the domain modeling process using the DoSS framework.

4.3.2.2 Preliminary Domain Model

The DoSS framework follows the following steps:

**Identify all entities and their values in the scenario.** Routinely, this includes all the nouns used in the definition of the scenario.

NL01 has been reproduced below with all entities and values highlighted in bold text:

*Aircraft ER-1357 has departed from the Daytona Beach International Airport (KDAB) via runway 7L. ER-1357 will be landing on runway 29 at the Gainesville Regional Airport (KGNV). Stable weather conditions along the route are reported as calm winds, dew point: 52, sky condition: scattered clouds at 5500 feet, temperature: 65, visibility: 10. Once within 30 miles of KGNV, Jacksonville approach control contacts ER-1357 and the landing phase begins. The pilot of ER-1357 will need to contact ATC for descent, approach, and landing clearances and will land normally without any complications.*
Regional Airport (KGNV). Stable weather conditions along the route are reported as calm winds, dew point: 52, sky condition: scattered clouds at 5500 feet, temperature: 65, visibility: 10. Once within 30 miles of KGNV, Jacksonville approach control contacts ER-1357 and the landing phase begins. The pilot of ER-1357 will need to contact ATC for descent, approach, and landing clearances and will land normally without any complications.

These entities and their values have been recorded in Table 1.

Table 1: List of entities and values identified

<table>
<thead>
<tr>
<th>No.</th>
<th>Entity</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aircraft</td>
<td>ER-1357</td>
</tr>
<tr>
<td>2</td>
<td>Airport</td>
<td>Daytona Beach International Airport (KDAB), Gainesville Regional Airport (KGNV)</td>
</tr>
<tr>
<td>3</td>
<td>Runway</td>
<td>7L, 29</td>
</tr>
<tr>
<td>4</td>
<td>Weather Conditions</td>
<td>Stable</td>
</tr>
<tr>
<td>5</td>
<td>Winds</td>
<td>Calm</td>
</tr>
<tr>
<td>6</td>
<td>Dew Point</td>
<td>52</td>
</tr>
<tr>
<td>7</td>
<td>Sky Condition</td>
<td>Few clouds at 5500 feet</td>
</tr>
<tr>
<td>8</td>
<td>Temperature</td>
<td>65</td>
</tr>
<tr>
<td>9</td>
<td>Visibility</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>ATC</td>
<td>Jacksonville Approach Control</td>
</tr>
<tr>
<td>11</td>
<td>Pilot</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>Clearance</td>
<td>Descent, Approach, Landing</td>
</tr>
<tr>
<td>13</td>
<td>Normal Landing</td>
<td>-</td>
</tr>
</tbody>
</table>
**Determine the interactions between these entities.** Interactions are usually defined by the use of a verb or a preposition connecting two nouns, either entities or their specific values in the scenario.

NL01 has been reproduced below with all interactions highlighted in italics, and the entities that interact with each other highlighted in bold text:

**Aircraft ER-1357 has departed from the Daytona Beach International Airport (KDAB) via runway 7L. ER-1357 will be landing on runway 29 at the Gainesville Regional Airport (KGNV).** Stable weather conditions along the route are reported as calm winds, dew point: 23, sky condition: scattered clouds at 5500 feet, temperature: 65, visibility: 10. Once within 30 miles of KGNV, *Jacksonville approach control contacts ER-1357* and the landing phase begins. The *pilot of ER-1357* will need to *contact ATC for descent, approach, and landing clearances* and will land normally without any complications.

These entities and their interactions have been recorded in Table 2.

**Table 2: List of entities and interactions**

<table>
<thead>
<tr>
<th>No.</th>
<th>Entity</th>
<th>Value</th>
<th>Interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aircraft</td>
<td>ER-1357</td>
<td>Airport, Runway, ATC, Pilot</td>
</tr>
<tr>
<td>2</td>
<td>Airport</td>
<td>Daytona Beach International Airport (KDAB),</td>
<td>Aircraft, Runway</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gainesville Regional Airport (KGNV)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Runway</td>
<td>7L, 29</td>
<td>Airport, Aircraft</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---------------</td>
</tr>
<tr>
<td>4</td>
<td>Weather Conditions</td>
<td>Stable</td>
<td>Winds, Dew Point, Sky Condition, Temperature, Visibility</td>
</tr>
<tr>
<td>5</td>
<td>Winds</td>
<td>Calm</td>
<td>Weather Conditions</td>
</tr>
<tr>
<td>6</td>
<td>Dew point</td>
<td>52</td>
<td>Weather conditions</td>
</tr>
<tr>
<td>7</td>
<td>Sky condition</td>
<td>Few clouds at 5500 feet</td>
<td>Weather Conditions</td>
</tr>
<tr>
<td>8</td>
<td>Temperature</td>
<td>65</td>
<td>Weather Conditions</td>
</tr>
<tr>
<td>9</td>
<td>Visibility</td>
<td>10</td>
<td>Weather Conditions</td>
</tr>
<tr>
<td>10</td>
<td>ATC</td>
<td>Jacksonville Approach Control</td>
<td>Aircraft, Clearance</td>
</tr>
<tr>
<td>11</td>
<td>Pilot</td>
<td>-</td>
<td>Aircraft, ATC</td>
</tr>
<tr>
<td>12</td>
<td>Clearance</td>
<td>Descent, Approach, Landing</td>
<td>ATC, Aircraft</td>
</tr>
</tbody>
</table>

**Define elements based on interactions.** Ideally, any entity directly interacting with more than one other entity should be a parent entity or a model object. An entity that is connected to just one other entity can be considered either a model object or an object’s attribute. This definition may change as new scenarios are considered during the design process.

Based on the list of elements in Table 2, model objects and attributes need to be determined. The entities associated with more than one other entity will automatically be considered as model objects. These are Aircraft, Airport, Runway, Weather conditions, ATC, Pilot, and Clearance. The entities which are associated with one other entity may be considered as either model objects or attributes of that object. These are Winds, Dew point, Sky condition, Temperature, and Visibility. As all of these elements only have a single value and no other associations, they will be considered as attributes of the object.
“Weather conditions” based on this scenario. The final list of elements and attributes based on this scenario can be seen in Table 3.

<table>
<thead>
<tr>
<th>No.</th>
<th>Element</th>
<th>Attributes</th>
<th>Associated with</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aircraft</td>
<td>ID</td>
<td>Airport, Runway, ATC, Pilot</td>
</tr>
<tr>
<td>2</td>
<td>Airport</td>
<td>ID, Name</td>
<td>Aircraft, Runway</td>
</tr>
<tr>
<td>3</td>
<td>Runway</td>
<td>ID</td>
<td>Airport, Aircraft</td>
</tr>
<tr>
<td>4</td>
<td>Weather Conditions</td>
<td>Type</td>
<td>Winds, Dew Point, Sky Condition, Temperature, Visibility</td>
</tr>
<tr>
<td>5</td>
<td>ATC</td>
<td>ID</td>
<td>Aircraft, Clearance</td>
</tr>
<tr>
<td>6</td>
<td>Pilot</td>
<td>ID</td>
<td>Aircraft, ATC</td>
</tr>
<tr>
<td>7</td>
<td>Clearance</td>
<td>Type</td>
<td>ATC, Aircraft</td>
</tr>
</tbody>
</table>

**Combine all elements and interactions.** Each entity’s interaction with all other entities is drawn to create the model centering on that particular entity. All these entity diagrams are then combined to obtain the domain model based on the given scenario. A sample of scenarios that make up the requirement model should contain all the necessary information about the domain model, and combining these domain models for all scenarios should provide the model for the DSL.
These entities and their interactions are first drawn up separately for each entity in Table 3. These figures show the representation of each entity in the following figures. Aircraft: Figure 11, Airport: Figure 12, Runway: Figure 13, ATC: Figure 14, Pilot: Figure 15, and Clearance: Figure 16. The Weather Conditions entity has not been shown as it only has attributes, but no entities associated with it.

**Figure 11: Representation of Aircraft entity model**

**Figure 12: Representation of Airport entity model**

**Figure 13: Representation of Runway entity model**
All these entity models are combined to obtain the preliminary domain model based on the given scenario. A parent entity called “Scenario” is added to include all other entities in the scenario. The domain model based on NL01 is shown in Figure 17.
Figure 17: Combined domain model based on Scenario NL01

This has been modeled using EMF to show all base objects and attributes and can be seen in Figure 18.

Figure 18: Preliminary domain model based on scenarios
4.3.2.3 Preliminary Formal Model

A preliminary formal model can be constructed using the scenario narrative as an indicator of changes made to domain entities. In this case, as the entity with the most associations and changes of interest is Aircraft, a statechart will be created for this entity.

Table 4 shows the changes in the state of an aircraft based on the scenario description for NL01, reproduced here for convenience.

Aircraft ER-1357 has departed from the Daytona Beach International Airport (KDAB) via runway 7L. ER-1357 will be landing on runway 29 at the Gainesville airport (KGNV). Stable weather conditions along the route are reported as calm winds, dew point: 23, sky condition: scattered clouds at 5500 feet, temperature: 15, visibility: 10.
Once within 30 miles of KGNV, Jacksonville approach control contacts ER-1357 and the landing phase begins. The pilot of ER-1357 will need to contact ATC for descent, approach, and landing clearances and will land normally without any complications.

Table 4: List of actions and states identified from Scenario NL01

<table>
<thead>
<tr>
<th>Action</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request descent clearance</td>
<td>Descent (if received)</td>
</tr>
<tr>
<td>Request approach clearance</td>
<td>Approach (if received)</td>
</tr>
<tr>
<td>Request landing clearance</td>
<td>Land (if received)</td>
</tr>
</tbody>
</table>
This list lends itself to the creation of a simple statechart, where states change as long as the actions are completed, with no possibilities of other actions or states in between. In this case, the initial state is chosen to be “Departed”, as that is the first thing the scenario mentions about the aircraft, with no information provided about the action that led to that state. Since nothing is mentioned following landing, it is considered the final state of the aircraft at the conclusion of this scenario. This preliminary statechart can be seen in Figure 19.

![Figure 19: Preliminary statechart based on Scenario NL01](image)

### 4.3.2.4 Scenarios 2-5

As the domain model and statechart obtained in the previous steps are very limited, a larger set of scenarios is required in order to depict all aspects of the domain. The scenarios used in order to do this have been described in this section. The same steps were followed for each scenario to lead to a domain model in each case.

1. Scenario 2, Normal Landing, NL02

*Aircraft ER-2101 has departed from the Daytona Beach International Airport (KDAB) via runway 7L. ER-2101 will be landing on runway 29 at the Gainesville airport (KGNV). Stable weather conditions along the route are reported as calm winds, dew point: 54, sky condition: clear, temperature: 65, visibility: 10. Once within 30 miles of KGNV, Jacksonville approach control contacts ER-2101 and the landing phase begins.*
Pilot John Smith of ER-2101 will need to contact ATC for descent, approach, and landing clearances and will perform a normal landing. The scenario ends when ER-2101 is safely parked at Gate B17 after taxiing.

2. Scenario 3, Normal Landing, NL03

Aircraft ER-2212 is cruising at 28000 ft after departing from the Daytona Beach International Airport (KDAB) via runway 7L. ER-2212 will be landing on runway 29 at the Hartsfield-Jackson Atlanta International Airport (KATL). Once within 30 miles of KATL, Atlanta approach control contacts ER-2212 and the landing phase begins. The pilot contacts ATC for descent and approach clearances which are granted, but upon requesting landing clearances, ER-2212 is asked to enter a holding pattern and await further clearance. The clearances are issued again and the aircraft will land normally without any complications.

3. Scenario 4, Crosswind Landing, CW01

Aircraft ER-1703 is cruising at 28000 ft after departing from the Daytona Beach International Airport (KDAB) via runway 7L. ER-1703 will be landing on runway 29 at the Hartsfield-Jackson Atlanta International Airport (KATL). Once within 30 miles of KATL, Atlanta approach control contacts ER-1703 and the landing phase begins. The pilot contacts ATC for descent, approach, and landing clearances, but while these are granted, the pilot is informed that there are high crosswinds of 25 knots. The pilot needs to account for this by adjusting flight properties such as flap settings and ground speed and should perform a crosswind landing.
4. Scenario 5, Short Field Landing, SF01

*Aircraft* ER-3110 is cruising at 30000 ft after departing from the Daytona Beach International Airport (KDAB) via runway 16, headed towards Key West International Airport (KEYW). 20 minutes after takeoff, ER-3110 encounters an emergency and is required to land immediately at Sebring Regional Airport (KSEF). Clearances are issued immediately, but runway 19, where the aircraft is expected to land, is only 5200 feet long and the recommended length of the runway for the aircraft is 6500 feet. Pilot Jane Doe will have to manage the speed, angle, power, and flaps appropriately in order to perform a short field landing safely.

Further scenarios used have not been reproduced here as they are similar, with minor changes to parameters in order to capture all the forms of landing.

4.3.2.5 Domain Model

The newer model elements added based on the above scenario are types of landings (normal, crosswind, short field landings), a name for the pilot (which can replace the earlier-assigned ID), flight properties (such as flaps, ground speed, angle, and power), and a gate at the destination airport. Since the Aircraft state is the focus of the behavioral model, this is also added as a class. The new domain model can be seen in Figure 20.
Figure 20: Domain model in EMF using the scenario-based approach

4.3.2.6 Formal Model

The newer scenarios show the addition of a loop when landing clearance is not granted. There is also the addition of a new initial state: cruise, and a new final state: parked at gate. The extended formal model is shown in Figure 21.
4.3.3 Comparison

A comparison of the domain model and formal model is shown in this section to see if the use of the DoSS framework provides any advantages in the development of a domain model for a DSL.

4.3.3.1 Elements

As can be seen from Figure 10 and Figure 20, the same elements are ultimately present in the domain model irrespective of the approach. In this case, it took 5 scenarios to reach the same domain model as the one obtained using the ontology-based approach. This number was much smaller than expected as vastly different scenarios were used in order to capture all the possible aspects that might need to be modeled, but an average number of scenarios required cannot be predicted for any other type of model, or for any extension to this model.
4.3.3.2 Resources

The ontology-based model relied on the use of several external resources to capture the terminology and model elements required in the description of a flight. This was obtained from existing glossaries (Federal Aviation Administration, 2012; SESAR) by searching for terms relevant to an aircraft landing. Based on the definitions of these terms, they were separated into categories and placed in a hierarchical manner to complete the ontology. The terms and organization of this ontology were then used to create the domain model (Jafer, Chhaya, & Durak, 2017), which was then used to describe scenarios.

In the DoSS-based approach, the scenarios that would ultimately need to be modeled were the only artifacts used during the design process. The scenarios were studied in order to determine the entities, attributes, and their interactions for a single scenario at a time, and an incremental domain model was created using these scenario diagrams at each stage. This required each scenario narrative to be studied deeply in order to extract these details, which took an appreciable amount of time, but still lesser than looking for external sources of information and compiling them to create an ontology before starting the process of creating the domain model.

4.3.3.3 Summary

A comparison of the two approaches shows a difference in three major areas: time and effort required, prior knowledge required, and formal artifacts produced. A scenario-based approach takes lesser time and effort than the ontology-based approach as it does not require the creation of a separate formally defined ontology. The time taken is
somewhat increased by the need to have well-defined scenarios that need to be parsed multiple times manually in order to capture all the domain requirements that are present. It also does not require prior expertise in the domain of interest, but rather just the presence of well-defined scenarios which need to be represented by the model. However, an ontology is a formally defined set of keywords and relationships which makes the basis of the model logically sound and verifiable when used to build a domain model.

Some key differences between the two approaches have been highlighted in Table 5. It can be noted based on this information that a scenario-based approach would be highly desirable if there is a formal process developed for it, as it can save time and reduce the assistance required from a domain expert.

Table 5: Comparison of DSL modeling approaches

<table>
<thead>
<tr>
<th>Element</th>
<th>Ontology</th>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Documentation</strong></td>
<td>Requires the creation of a new formally defined artifact that captures domain concepts and behavior.</td>
<td>Requires the presence of well-defined scenarios that are a part of requirements and testing documentation.</td>
</tr>
<tr>
<td><strong>Domain</strong></td>
<td>Requires a domain expert to define and describe the concepts and relationships in the domain prior to domain modeling.</td>
<td>Requires only the information captured within the scenario for domain modeling.</td>
</tr>
<tr>
<td><strong>Language</strong></td>
<td>There are no language constraints as all information is</td>
<td>The scenarios need to be well-defined and deliberately</td>
</tr>
<tr>
<td>Time</td>
<td>The creation of an ontology requires additional time as all domain information needs to be aggregated and described.</td>
<td>Additional time is taken in multiple reading of scenario descriptions in order to capture all available domain information.</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Process</td>
<td>Ontologies are formal representations of domain knowledge and there are strict rules about their creation as well as their transformation into domain models.</td>
<td>Scenarios are informal/natural language descriptions of events and are used as a basis for defining requirements and test cases. There is currently no formal process for converting such natural language scenarios into domain models.</td>
</tr>
</tbody>
</table>

As the two domain models at this stage (for modeling aircraft landing) were shown to be very similar, the biggest factor in choosing a methodology is the amount of resources and time consumed. It was clear that fewer sources of information were required using DoSS, but the time taken was still significant as it required the developer to go over each scenario narrative multiple times in order to extract all the elements required from it. It
also required a basic grasp of the English language and the ability to distinguish different parts of speech. While a general guideline for identifying entities and interactions was provided on the basis of language construct in English, it still requires the developer to recognize grammatical anomalies in the scenario narrative if present. The DoSS framework is a promising alternative for developing a DSL without excessive domain knowledge, but it would be more valuable if the process of identification of keywords from scenario text could be automated to some extent. The next chapter discusses domain model development in more detail and presents a case study showing the use of this framework by someone without domain knowledge and experience.
Chapter 5

Comparison Case Study

This chapter discusses a case study that attempted to answer the third research question:

*What value does the implementation of DSLs using only elicited requirements add in terms of a) time taken and b) technical knowledge required?*

For this study, an experiment was conducted with a human subject (user) tasked with creating a metamodel for aviation using only the scenarios presented here. The user is in the age range of 25-30 years old and is a graduate student in the Sciences. They have no prior experience in domain modeling, software development or software engineering practices. The user had no knowledge of flight training or ATC scenarios prior to seeing the DoSS instructions and had not seen an ontology or metamodel for it. The conducted experiment using the DoSS framework for two scenarios is shown here first, then the final metamodel is compared to the metamodel obtained by the author to see the value in the framework when used by someone who is not a domain or M&S expert. This case study is to demonstrate the usability of the DoSS framework when followed by someone who has no prior experience with either the domain or the M&S process, which was previously identified as a technical challenge in the literature review.
5.1 User-driven Scenario Analysis using DoSS Framework

5.1.1 Scenario 1

The first scenario analyzed by the user was as follows:

*Aircraft ER-2212 is cruising at 28000 ft after departing from the Daytona Beach International Airport (KDAB) via runway 7L. ER-2212 will be landing on runway 29 at the Hartsfield-Jackson Atlanta International Airport (KATL). Once within 30 miles of KATL, Atlanta approach control contacts ER-2212 and the landing phase begins. The pilot contacts Control for descent and approach clearances which are granted, but upon requesting landing clearances, ER-2212 is asked to enter a holding pattern and await further clearance. The clearances are issued again and the aircraft will land normally without any complications.*

5.1.1.1 Task 1

Task 1 was to identify all entities and their values in the scenario. The user found the entities listed in Table 6.

*Table 6: Entities and values obtained from Scenario 1 by the user*

<table>
<thead>
<tr>
<th>No.</th>
<th>Entity</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aircraft</td>
<td>ER-2212</td>
</tr>
<tr>
<td>2</td>
<td>Airport</td>
<td>Daytona Beach International Airport (KDAB), Hartsfield-Jackson Atlanta International Airport (KATL)</td>
</tr>
<tr>
<td>3</td>
<td>Runway</td>
<td>7L, 29</td>
</tr>
</tbody>
</table>
5.1.1.2 Task 2

Task 2 was to determine the interactions between these entities. These entities and their interactions have been recorded in Table 7.

Table 7: Entities and interactions obtained from Scenario 1 by the user

<table>
<thead>
<tr>
<th>No.</th>
<th>Entity</th>
<th>Value</th>
<th>Interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aircraft</td>
<td>ER-2212</td>
<td>Airport, Runway, Approach Control, Pattern, Clearance</td>
</tr>
<tr>
<td>2</td>
<td>Airport</td>
<td>Daytona Beach International Airport (KDAB), Hartsfield-Jackson Atlanta International Airport (KATL)</td>
<td>Runway, Aircraft, Approach Control</td>
</tr>
<tr>
<td>3</td>
<td>Runway</td>
<td>7L, 29</td>
<td>Airport, Aircraft</td>
</tr>
<tr>
<td>4</td>
<td>Approach Control</td>
<td>Atlanta</td>
<td>Airport, Aircraft, Clearance, Pattern</td>
</tr>
<tr>
<td>5</td>
<td>Pilot</td>
<td>-</td>
<td>Aircraft</td>
</tr>
</tbody>
</table>
5.1.1.3 Task 3

Task 3 was to define elements based on interactions. These elements as recorded by the user can be found in Table 8.

Table 8: Elements in the domain model based on Scenario 1 by the user

<table>
<thead>
<tr>
<th>No.</th>
<th>Element</th>
<th>Attributes</th>
<th>Associated with</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aircraft</td>
<td>ID</td>
<td>Airport, Runway, Approach Control, Pattern, Pilot, Clearance</td>
</tr>
<tr>
<td>2</td>
<td>Airport</td>
<td>ID, Name</td>
<td>Runway, Aircraft, Approach Control</td>
</tr>
<tr>
<td>3</td>
<td>Runway</td>
<td>ID</td>
<td>Airport, Aircraft</td>
</tr>
<tr>
<td>4</td>
<td>Approach Control</td>
<td>Name</td>
<td>Airport, Aircraft, Clearance, Pattern</td>
</tr>
<tr>
<td>5</td>
<td>Clearance</td>
<td>Type</td>
<td>Approach Control, Aircraft</td>
</tr>
<tr>
<td>6</td>
<td>Pattern</td>
<td>Type</td>
<td>Aircraft</td>
</tr>
</tbody>
</table>
5.1.1.4 Task 4

Task 4 was to create a domain model with the entities found in Tasks 1-3. The diagram based on the user’s domain model can be seen in Figure 22.

Figure 22: Domain model based on Scenario 1 by the user

5.1.1.5 Scenario Analysis Comparison

The same scenario, when analyzed by the author, had the properties described in Table 9-Table 11. The domain model is shown in Figure 23.

Table 9: Entities and values obtained from Scenario 1 by the author

<table>
<thead>
<tr>
<th>No</th>
<th>Entity</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aircraft</td>
<td>ER-2212</td>
</tr>
</tbody>
</table>
Table 10: Entities and interactions obtained from Scenario 1 by the author

<table>
<thead>
<tr>
<th>No.</th>
<th>Entity</th>
<th>Value</th>
<th>Interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aircraft</td>
<td>ER-2212</td>
<td>Airport, Runway, Approach, Control, Phase, Clearance, Pattern</td>
</tr>
<tr>
<td>2</td>
<td>Airport</td>
<td>Daytona Beach International Airport (KDAB), Hartsfield-Jackson Atlanta International Airport (KATL)</td>
<td>Aircraft, Runway, Approach Control</td>
</tr>
<tr>
<td>3</td>
<td>Runway</td>
<td>7L, 29</td>
<td>Aircraft, Airport</td>
</tr>
</tbody>
</table>
Table 11: Elements in the domain model based on Scenario 1 by the author

<table>
<thead>
<tr>
<th>No.</th>
<th>Entity</th>
<th>Value</th>
<th>Associated with</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aircraft</td>
<td>ID</td>
<td>Airport, Runway, Approach Control, Clearance,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pilot</td>
<td>Pattern</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Phase</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Airport</td>
<td>ID</td>
<td>Aircraft, Runway, Approach Control</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Runway</td>
<td>ID</td>
<td>Aircraft, Airport</td>
</tr>
<tr>
<td>4</td>
<td>Approach Control</td>
<td>Name</td>
<td>Aircraft, Airport, Clearance</td>
</tr>
<tr>
<td>5</td>
<td>Clearance</td>
<td>Type</td>
<td>Approach Control, Aircraft</td>
</tr>
</tbody>
</table>
5.1.2 Scenario 2

The second scenario analyzed by the user was:

*The route of flight follows well-established departure, climb, cruise, descent, and landing procedures. The procedures begin with a departure from runway 36 at Aurora Municipal Airport, with a climb out to 3000 feet. From midfield on the downwind leg, you will fly on a 225 magnetic heading until intercepting the JOLIET 290 radial. Then you will turn left to a heading of 210, continue climb to 4500 feet, and maintain that heading until intercepting the JOLIET 256 radial. Proceed inbound on the PONTIAC 348 radial, and descend down to 3500 feet. When over PONTIAC VOR, fly outbound on the PONTIAC*
182 radial until you intercept the BLOOMINGTON 060 radial, where you turn right for a 45-degree entry to enter a left-hand traffic pattern for landing on runway 11.

5.1.2.1 Task 1

Task 1 was to identify all entities and their values in the scenario. The user found the entities listed in Table 12.

Table 12: Entities and values obtained from Scenario 2 by the user

<table>
<thead>
<tr>
<th>No.</th>
<th>Entity</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Route</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Flight</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>Procedures</td>
<td>Departure, climb, cruise, descent, landing</td>
</tr>
<tr>
<td>4</td>
<td>Departure</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>Runway</td>
<td>36, 11</td>
</tr>
<tr>
<td>6</td>
<td>Airport</td>
<td>Aurora Municipal</td>
</tr>
<tr>
<td>7</td>
<td>Altitude</td>
<td>3000 ft, 4500 ft, 3500 ft</td>
</tr>
<tr>
<td>8</td>
<td>Heading</td>
<td>225 magnetic, 210</td>
</tr>
<tr>
<td>9</td>
<td>Radial</td>
<td>JOLIET 290, JOLIET 256, PONTIAC 348, PONTIAC VOR, PONTIAC 182, BLOOMINGTON 060</td>
</tr>
<tr>
<td>No.</td>
<td>Entity</td>
<td>Value</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>10</td>
<td>Entry</td>
<td>45 degree</td>
</tr>
<tr>
<td>11</td>
<td>Pattern</td>
<td>Left-hand traffic</td>
</tr>
<tr>
<td>12</td>
<td>Landing</td>
<td>-</td>
</tr>
</tbody>
</table>

The entities and values obtained by the author for the same scenario can be seen in Table 13.

Table 13: Entities and values obtained from Scenario 2 by the author

<table>
<thead>
<tr>
<th>No.</th>
<th>Entity</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Route</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Flight</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>Procedures</td>
<td>Departure, Climb, Cruise, Descent, Landing</td>
</tr>
<tr>
<td>4</td>
<td>Departure</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>Runway</td>
<td>36, 11</td>
</tr>
<tr>
<td>6</td>
<td>Airport</td>
<td>Aurora Municipal</td>
</tr>
<tr>
<td>7</td>
<td>Altitude</td>
<td>3000 ft, 4500 ft, 3500 ft</td>
</tr>
<tr>
<td>8</td>
<td>Heading</td>
<td>225, 210</td>
</tr>
<tr>
<td>9</td>
<td>Radial</td>
<td>JOLIET 290, JOLIET 256, PONTIAC 348, PONTIAC 182, BLOOMINGTON 060</td>
</tr>
</tbody>
</table>
5.1.2.2 Task 2

Task 2 was to determine the interactions between these entities. These entities and their interactions have been recorded in Table 14.

Table 14: Entities and interactions obtained from Scenario 2 by the user

<table>
<thead>
<tr>
<th>No.</th>
<th>Entity</th>
<th>Value</th>
<th>Interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Route</td>
<td>-</td>
<td>Flight, procedures</td>
</tr>
<tr>
<td>2</td>
<td>Flight</td>
<td>-</td>
<td>Route, procedures, departure, runway, airport, altitude, heading, radial, entry, pattern, landing</td>
</tr>
<tr>
<td>3</td>
<td>Procedures</td>
<td>Departure, climb, cruise, descent, landing</td>
<td>Route</td>
</tr>
<tr>
<td>4</td>
<td>Departure</td>
<td>-</td>
<td>Airport, flight, runway</td>
</tr>
</tbody>
</table>
The entities and values obtained by the author for the same scenario can be seen in Table 15.

Table 15: Entities and interactions obtained from Scenario 2 by the author

<table>
<thead>
<tr>
<th>No.</th>
<th>Entity</th>
<th>Value</th>
<th>Interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Runway</td>
<td>36,11</td>
<td>Airport, flight, departure,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>landing</td>
</tr>
<tr>
<td>6</td>
<td>Airport</td>
<td>Aurora Municipal</td>
<td>Runway, flight, departure,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>landing</td>
</tr>
<tr>
<td>7</td>
<td>Altitude</td>
<td>3000 ft, 4500 ft, 3500 ft</td>
<td>Flight</td>
</tr>
<tr>
<td>8</td>
<td>Heading</td>
<td>225 magnetic, 210</td>
<td>Flight</td>
</tr>
<tr>
<td>9</td>
<td>Radial</td>
<td>JOLIET 290, JOLIET 256, PONTIAC 348, PONTIAC VOR,</td>
<td>Flight</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PONTIAC 182, BLOOMINGTON 060</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Entry</td>
<td>45 degree</td>
<td>Flight, pattern</td>
</tr>
<tr>
<td>11</td>
<td>Pattern</td>
<td>Left-hand traffic</td>
<td>Entry, flight, landing</td>
</tr>
<tr>
<td>12</td>
<td>Landing</td>
<td>-</td>
<td>Flight, airport, runway</td>
</tr>
<tr>
<td>2</td>
<td>Flight</td>
<td>-</td>
<td>Route, Procedures, Departure, Runway, Airport, Altitude, Heading, Radial, VOR, Turn, Entry, Pattern, Landing</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>3</td>
<td>Procedures</td>
<td>Departure, Climb, Cruise, Descent, Landing</td>
<td>Route, Flight</td>
</tr>
<tr>
<td>4</td>
<td>Departure</td>
<td>-</td>
<td>Airport, Flight, Runway</td>
</tr>
<tr>
<td>5</td>
<td>Runway</td>
<td>36, 11</td>
<td>Airport, Flight, Departure, Landing</td>
</tr>
<tr>
<td>6</td>
<td>Airport</td>
<td>Aurora Municipal</td>
<td>Runway, Flight, Departure, Landing</td>
</tr>
<tr>
<td>7</td>
<td>Altitude</td>
<td>3000 ft, 4500 ft, 3500 ft</td>
<td>Flight</td>
</tr>
<tr>
<td>8</td>
<td>Heading</td>
<td>225, 210</td>
<td>Flight</td>
</tr>
<tr>
<td>9</td>
<td>Radial</td>
<td>JOLIET 290, JOLIET 256, PONTIAC 348, PONTIAC 182, BLOOMINGTON 060</td>
<td>Flight</td>
</tr>
<tr>
<td>10</td>
<td>VOR</td>
<td>PONTIAC</td>
<td>Flight</td>
</tr>
<tr>
<td>11</td>
<td>Turn</td>
<td>Right</td>
<td>Flight</td>
</tr>
<tr>
<td>12</td>
<td>Entry</td>
<td>45 degree</td>
<td>Flight, Pattern</td>
</tr>
</tbody>
</table>
5.1.2.3 Task 3

Task 3 was to define elements based on interactions. These elements as recorded by the user can be found in Table 16.

Table 16: Elements in the domain model based on scenario 2 by the user

<table>
<thead>
<tr>
<th>No.</th>
<th>Element</th>
<th>Attributes</th>
<th>Associated with</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Route</td>
<td>-</td>
<td>Flight, procedures</td>
</tr>
<tr>
<td>2</td>
<td>Flight</td>
<td>Altitude, Heading, Radial</td>
<td>Route, procedures, departure, runway, airport, entry, pattern, landing</td>
</tr>
<tr>
<td>3</td>
<td>Procedures</td>
<td>Type</td>
<td>Route</td>
</tr>
<tr>
<td>4</td>
<td>Departure</td>
<td>-</td>
<td>Airport, flight, runway</td>
</tr>
<tr>
<td>5</td>
<td>Runway</td>
<td>ID</td>
<td>Airport, flight, departure, landing</td>
</tr>
<tr>
<td>6</td>
<td>Airport</td>
<td>Name</td>
<td>Runway, flight, departure, landing</td>
</tr>
<tr>
<td>7</td>
<td>Entry</td>
<td>Angle</td>
<td>Flight, pattern</td>
</tr>
</tbody>
</table>
The same task completed by the author yields the values shown in Table 17.

Table 17: Elements in the domain model based on scenario 2 by the author

<table>
<thead>
<tr>
<th>No.</th>
<th>Element</th>
<th>Attributes</th>
<th>Associated with</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Route</td>
<td>-</td>
<td>Flight, Procedures</td>
</tr>
<tr>
<td>2</td>
<td>Flight</td>
<td>Altitude, Heading, Radial, VOR, Turn</td>
<td>Route, Procedures, Departure, Runway, Airport, Entry, Pattern, Landing</td>
</tr>
<tr>
<td>3</td>
<td>Procedures</td>
<td>Type</td>
<td>Route, Flight</td>
</tr>
<tr>
<td>4</td>
<td>Departure</td>
<td>-</td>
<td>Airport, Flight, Runway</td>
</tr>
<tr>
<td>5</td>
<td>Runway</td>
<td>ID</td>
<td>Airport, Flight, Departure, Landing</td>
</tr>
<tr>
<td>6</td>
<td>Airport</td>
<td>Name</td>
<td>Runway, Flight, Departure, Landing</td>
</tr>
</tbody>
</table>
5.1.1.4 Task 4

Task 4 was to create a domain model with the entities found in Tasks 1-3. The diagram based on the user’s domain model can be seen in Figure 24.

![Domain model based on Scenario 2 by the user](image)

**Figure 24: Domain model based on Scenario 2 by the user**

The same task when performed by the author yielded the result shown in Figure 25.
5.2 User Domain Model using DoSS Framework

Eight other relevant scenarios were analyzed by the user to obtain a final domain model. The work for the remaining scenarios has not been shown in the interest of brevity. At the conclusion of scenario analysis, Task 5 was to create a final domain model including all the entities that were obtained from the scenario and all the connections between them. The domain model obtained by the user has been shown in Figure 26.
The same tasks were performed by the author. The domain model obtained by the author is seen in Figure 27.

It can be seen that the models are very similar to each other. The next chapter analyzes the models obtained and the value of the framework in the domain modeling process.
Chapter 6

Analysis

This chapter discusses the results from the case study presented in Chapter 5 and provides an overall analysis of the DoSS framework for developing DSLs. The metrics used to determine the usability of this framework from a non-domain expert point-of-view are:

1. Number of differences between user domain model and author domain model,
2. Time taken to create the model,
3. Effort required to create the model, and
4. Number of sample scenarios defined by the model.

A discussion of model behavior verification using statecharts follows after performing the case study analysis. The validation and verification of the final models by the user and author were performed by domain experts to determine the applicability of the methodology in the aviation domain.

6.1 **Number of differences between user domain model and author domain model**

The same ten scenarios were used in the creation of the domain model by both the user and the author. The final domain model diagrams are hard to compare directly due to the large number of entities and connections present in each. This section compares the classes and attributes individually, so it is easier to see where the models differ. Figure 28
shows the top-level entities in the two domain models. Figure 28 (a) shows the user’s domain model and Figure 28 (b) shows the author’s model.

![Figure 28: All top-level entities for the two domain models](image)

As seen in Figure 28, the user’s domain model has 18 main entities and the author’s domain model has 20 entities. The author has an additional phase entity (“Climb → Phase”) and an additional route entity (“STAR → Route”), but the other 18 entities are identical. The attributes and connections included within these entities have been further broken down in Figure 29 to Figure 33. Sub-figure (a) shows the user’s domain model and sub-figure (b) shows the author’s domain model in the following figures.
Figure 29: Attributes and references for the two domain models – 1/5

Figure 30: Attributes and references for the two domain models – 2/5
Figure 31: Attributes and references for the two domain models – 3/5

Figure 32: Attributes and references for the two domain models – 4/5
Based on the figures above, the difference in the elements in the two domain models has been listed in Table 18.

Table 18: Differences in the two domain models

<table>
<thead>
<tr>
<th>Type</th>
<th>No. in user’s model</th>
<th>No. in author’s model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entities</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>Attributes</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>References</td>
<td>41</td>
<td>41</td>
</tr>
</tbody>
</table>

The differences in the entities have already been described. While the number of attributes is identical, there is one difference. The user has STAR as an attribute, which the author has it as an entity. The author has an attribute for turning under FlightProperties, which the user is missing. The references between the classes are identical in both the models. Overall, it can be said that the two models are almost identical.
6.2 Time taken to create the model

A listed advantage of the DoSS framework is that it requires much less time commitment to create a domain model as compared to the current process that follows ontology development. While this is not a metric of comparison, this case study aims to set a baseline for time consumption in the process, so that future case studies and experiments can compare the time taken by a non-expert in creating a domain model.

An example of the time log filled out by the user for each scenario can be seen in Table 19. The time taken by the user in this case study (in minutes) can be seen in Figure 34.

Table 19: Time log filled out by user for Scenario 1

<table>
<thead>
<tr>
<th>Task</th>
<th>Time Spent (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reading Instructions</td>
<td>3</td>
</tr>
<tr>
<td>Reading Scenario</td>
<td>2</td>
</tr>
<tr>
<td>Step 1: Identify entities and values</td>
<td>2</td>
</tr>
<tr>
<td>Step 2: Determine interactions between entities</td>
<td>3</td>
</tr>
<tr>
<td>Step 3: Define elements based on interactions</td>
<td>2</td>
</tr>
<tr>
<td>Step 4: Combine all elements and interactions</td>
<td>3</td>
</tr>
<tr>
<td><strong>Total Time Spent</strong></td>
<td><strong>15</strong></td>
</tr>
</tbody>
</table>
The effort to parse all ten scenarios took the user 19.2 minutes on average per scenario and a total of 192 minutes (3.2 hours). The effort to create the domain model after obtaining all entities and interactions took another 75 minutes, for a total of 267 minutes (approximately 4.5 hours). While there is no recorded data as the author did not create the ontology-based model in a controlled setting, it is to be reasonably expected that a domain model following the ontology-based approach would take much longer.

6.3 Effort required to create the model

The effort required to create the model is measured in the form of the NASA Task Load Index (NASA-TLX) Subjective Workload Measure.
6.3.1 NASA-TLX Subjective Workload Measure

There are many ways of measuring workload, including physiological measures such as hormonal levels, pupil diameter, heart rate, sweat rate, etc.; however, many of those measures do not capture subjective workload as reported by the users themselves. The NASA-TLX was developed at NASA's Ames Research Center by the Human Performance (Hart & Staveland, 1988; Hart, 1986). This workload measure was developed precisely for the purpose of measuring user subjective workload and is useful for a variety of tasks, including online or computer-based training sessions. This multidimensional assessment tool rates users’ perceived workload, which is then correlated with other aspects of performance such as accuracy, speed, response times, etc. It is one of the most well-known and used self-report workload measures (Moroney, Biers, & Eggemeier, 1995; Noyes & Bruneau, 2007). The psychometric characteristics of the NASA-TLX are well documented (Yurko, Scerbo, Prabhu, Acker, & Stefanidis, 2010) and it has been used previously as a tool for subjective evaluation of an individual's workload in flight simulation (Nygren, 1991) and air traffic control studies (Metzger & Parasuraman, 2005).

The NASA-TLX is composed of two parts. In the first part, participants respond to six subscales that are presented on a single page. These subscales include:

- **Mental Demand.** What is the required level of mental and perceptual activity? How easy, difficult, simple, or complex was the task?

- **Physical Demand.** What is the required level of physical activity? How slack or strenuous was the task?
• **Temporal Demand.** How much time pressure is felt by the user? Was the pace too slow or too fast?

• **Performance.** How well does the user feel that she or he did on the task? How satisfied is the user with his or her performance?

• **Effort.** How hard did the user need to work in order to accomplish the task? This can be both physical and mental.

• **Frustration.** How much irritation, stress, or annoyance was perceived by the user?

Prior to responding to the scale questions, participants read the description for each subscale. They then provide a score for each subscale by choosing one of the gradations that range from Very Low to Very High. The scores can range from 0 to 100 in 5-point gradations. Upon completion of the NASA-TLX, user scores are then combined into an overall score that measures perceived subjective workload. Figure 35 provides the actual survey that is given to users.

As with all measures of workload, and particularly with subjective workload, there are some caveats. First, the NASA-TLX relies on users accurately giving their responses, and it assumes that the users are being honest with themselves and with the experimenter. Second, it relies on the users’ memory in order to accurately assess their workload. This can be problematic, especially if the survey is given some time after the task has ended. Third, the survey cannot be given while a user is doing the task in question, or else it becomes a dual-task situation. This can be problematic if the user suddenly feels a release of stress and frustration when the task is placed to the side while they fill out the survey. They may misperceive this reduction of stress as being part of the task and provide an inaccurate assessment of their subjective workload.
In this case study, the NASA-TLX was given to the user immediately after each scenario, and not just at the end of the overall session. The user was carefully instructed to rate their perceived workload during the task itself and not their perceived workload of taking the survey. The user was also encouraged to provide honest and well-thought-out responses in order to ensure validity.

Figure 35: NASA Task Load Index
(Hart, 1986)
6.3.2 Subjective Workload for DoSS Domain Modeling

The user was asked to fill out the NASA-TLX at the end of each scenario and once after finishing all the tasks. A snapshot of the results shared by the user can be seen in Table 20. A graph of the average effort for each metric can be seen in Figure 36.

Table 20: Snapshot of NASA-TLX responses from the user for scenario parsing using the DoSS framework

<table>
<thead>
<tr>
<th></th>
<th>Scenario 1</th>
<th>Scenario 2</th>
<th>Scenario 3</th>
<th>Scenario 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mental Demand</td>
<td>40</td>
<td>40</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Physical Demand</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Temporal Demand</td>
<td>40</td>
<td>35</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>Effort</td>
<td>45</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>Performance</td>
<td>85</td>
<td>85</td>
<td>90</td>
<td>85</td>
</tr>
<tr>
<td>Frustration</td>
<td>15</td>
<td>20</td>
<td>10</td>
<td>15</td>
</tr>
</tbody>
</table>
Lower scores indicate a lower perceived workload on the NASA-TLX scale. For this purpose, the performance score is changed to its complement out of 100 (100 – raw score), so it is shown to be 15 in Figure 36 compared to the raw score of 85 in Table 20. In this case study, all average values are below 50, showing that the user did not perceive their workload to be very high. The scores for mental demand ranged from 20 (scenario 5) to 60 (scenarios 9 and 10) and effort ranged from 30 (scenario 5) to 70 (scenario 9). The physical demand was always zero because there was no physical exertion required in this experiment. The scores for temporal demand, performance, and frustration were fairly consistent across the scenarios, ranging from 30 to 40 for temporal demand, 80 to 90 for performance, and 10 to 20 for frustration.
Overall, these are low numbers that show that following the DoSS framework is not very demanding for a non-expert in the domain.

A quantitative evaluation of the user’s workload was performed above. They were then asked for a qualitative opinion on the process and their feelings about the tasks. Their feedback has been reproduced below without edits.

Identifying proper and common nouns and then listing them felt like a slightly challenging word puzzle to me. I did not feel that I needed domain knowledge for classifying nouns in any scenario, and the first step was fairly easy. The identification of relationships between entities was a little more challenging, but nothing very strenuous. Based on whether an entity was interacting with one or multiple other entities, the listing of parent entities followed.

The time taken and effort expended obviously varied between different scenarios, with the identification of relationships taking longer, and requiring more mental work in longer scenarios with many entities. Overall, I did not feel too stressed even while doing the longer scenarios, and I did not feel the time required was excessive in any way.

The combination of entities to create a final model did take some time and effort, but nothing extraordinarily taxing. I think the relationships were mostly straightforward, and I was confident about the correctness of my final models.

Both the qualitative and quantitative responses from the user were positive with respect to the workload involved.
6.4 Number of sample scenarios defined by the model

In order to substantiate the claim that the DoSS framework allows non-experts to create domain models using only natural language scenarios, the user’s domain model was tested for effectiveness by attempting to use it to describe other scenarios. Ten scenarios were used to create the original domain model and then five new scenarios were checked against this model.

Two scenarios have been described in this section.

6.4.1 Test Scenario 1

The flight begins with a departure from runway 9 at Riverside Airport, with a climb out to 4500 feet. After departure, you will be turning to a heading of 256 to intercept the POMONA 164 radial. You will be tracking inbound on the 164 radial, and performing a step descent to set up for a right downwind approach to La Verne’s 26R (right) runway.

The elements in this scenario are flight, departure, runway, airport, heading, radial, descent, and approach. All these elements can be seen in Figure 28 with the appropriate attributes and connections, so it can be seen that this scenario is described by the domain model created by the user.

6.4.2 Test Scenario 2

This route of flight follows well-established departure, climb, cruise, descent, and landing procedures. You will depart Port Angeles, Fairchild International Airport at a departure time of 2230 Zulu, on runway 31. Climb up to and maintain 9000 feet as a cruise altitude, where along the way you’ll be handed off to Seattle Center. Your true speed will be close to 130 knots. Then you fly inbound on the Seattle VOR 228 radial. By
now, you’ll be under the jurisdiction of Seattle Approach Control. The expectation is to
descend to 3000 feet for the Mall Visual Approach Runway 34R into Seattle-Tacoma
International.

The elements in this scenario are route, flight, procedures, airport, time, runway, altitude,
center, speed, radial, approach control, and approach. All these entities can be seen in
Figure 28 with the appropriate attributes and connections, so it can be seen that this
scenario is described by the domain model created by the user.

All five scenarios can be described using the domain model at this stage, showing that the
domain model has some value for describing scenarios in the domain.

6.5 Verification of model behavior

The scenarios used for the final version of the domain model were used to iteratively
create a statechart to describe the properties of a flight over time and its interactions with
external entities such as ATC.

The process for the creation of this statechart followed the same steps as discussed in the
DoSS Framework section (Section 4.3.2). There is an existing method that uses natural
language scenarios for verification of software systems called the Method for SCENario-
Based Validation and Test of Software (SCENT Method) (Ryser & Glinz, 1999; Ryser &
Glinz, Using dependency charts to improve scenario-based testing, 2000; Ryser & Glinz,
SCENT: A method employing scenarios to systematically derive test cases for system
test, 2000), so the rationale for the use of this method for verification of DSLs has been
described in the literature review (Section 2.3).
However, the DoSS framework validation process builds upon the SCENT method and differs from it in two ways:

1. The SCENT method creates a statechart for each scenario whereas DoSS uses an iterative process to generate one statechart that describes the overall behavior of the system.

2. The focus of the SCENT method is to generate test cases from the statecharts whereas DoSS focuses on using statecharts to verify behavior described in natural language scenarios.

This statechart for the domain model created in the case study can be seen in Figure 37.

![Statechart](image)

**Figure 37: Statechart based on all scenarios by the user**

This statechart describes all the flight states and transitions described in the ten scenarios that were originally used to create the domain model and also models the other five scenarios that were used for verification in the previous section.
6.6 Validation and verification from domain experts

The case study and analysis of differences in the user and author’s work showed that it is possible for a non-expert in the domain to create a model similar to one created by someone with experience in ontology and domain modeling within the domain. However, this model ultimately needs to be evaluated by experts to understand its usability and value.

The domain model was first validated by attempting to generate executable scenarios using the ASDL EMF metamodel. These were generated based on natural language flight training scenarios and ATC training scenarios. Once it was determined that several scenarios were able to be defined, these scenarios were verified by domain experts. Specifically, verification of scenarios for flight training was done with the help of two licensed pilots who were involved as domain experts throughout the verification and validation process, and provided feedback on the domain model along with the scenarios generated.

ATC scenario verification was performed by two Air Traffic Management technical staff and instructors at the FAA Academy, with input from a Professor of Air Traffic Management in the Applied Aviation Sciences Department at the Daytona College of Aviation, Embry-Riddle Aeronautical University (ERAU).

The use of the NASA-TLX scale to quantify the effort required to use the DoSS framework was recommended by a Professor of Human Factors in the Human Factors and Behavioral Neurobiology Department at the Daytona College of Arts & Sciences,
ERAU. Their methodology for determining the effort was employed and an interpretation of the results was performed under their guidance.

The final model presented in this dissertation, both from the user and the author, were confirmed to be applicable by these domain experts and the scenarios generated using their properties have been employed in other projects including one developed for the FAA Academy, as will be discussed in Chapter 7.

While the scenarios were verified and approved by domain experts, the approach was presented to modeling and simulation experts at technical conferences with peer-reviewed paper presentations. This received positive feedback from reviewers and attendees. Some excerpts from reviewer comments are provided in this section.

The DoSS framework was presented at the AIAA SciTech Forum and Exposition in 2019 and was well-received there. The approach has also been peer-reviewed and presented at the Spring Simulation Conference 2020, hosted by the Society for Modeling & Simulation International. Some reviewer comments for this approach are:

1. “Scenarios are fetched from two different sources. For this study, I think this is sufficient. Of course, extending the dataset is better, as the authors state. I think it may be a good method to validate an ontology that has been already defined. Therefore, the ontology modelers can be sure that they did not miss any important elements. Some experiments can be done with some users that do not know an already developed ontology. Therefore, the contribution of the method can be more visible. More intelligent methods can be used to generate a model by using more sophisticated NLP tools.”
2. “Approach applied in this paper is almost new for the community and it seems there is a value. Of course to be sure of that, more samples (maybe from different sources) should be added to the work and then compare with the traditional methods. As discussed in conclusion, more samples will be useful. For this paper, these number of scenarios are enough to tell the approach. When more samples are ready, I have a recommendation about trying some Machine Learning approaches to extract requested information from scenario texts. Maybe that can be a future paper. Thank you again for your effort and new approach.”

The feedback from the reviewers was taken on board and shaped some tasks of this dissertation. The recommendation from Reviewer #1 to perform an experiment with a user with no domain ontology knowledge was the basis for the usability case study outlined in Chapter 5. Additional scenarios and sources were used for the case study based on the need pointed out by Reviewers #1 and #2. The recommendations from Reviewers #1 and #2 to include machine learning and natural language processing techniques were considered to be outside the scope of the project at this time, but have been listed under future work for this framework development.

The usability study as an example of the work that can be done by a non-expert has been submitted and accepted at the 2021 SciTech Forum and will be presented in January.

Additions have been made to the ASDL metamodel that now allow the formal specification of various types of aviation scenarios. ASDL was created with MSDL, which is a standard for military scenario specification, as a source of inspiration. Initial work was done towards applying for SISO standardization by defining ASDL using an XML schema that allows for format standardization and content verification (Jafer,
Chhaya, Updegrove, & Durak, 2018). The project has received feedback from aviation stakeholders for extensions to the model. The published XML schema of ASDL has been discussed by experts as part of the AIAA MSTC Working Group for Simulation Scenario Development.
Chapter 7

Applications and Extensions

This chapter discusses some applications and extensions of the research performed for this dissertation. This is broadly divided into two projects: (1) ATC Scenario Training Technology (ASTT), which is an application of ASDL, and (2) scenario-based ontology generation, which is an extension of the DoSS framework targeting ontology generation.

7.1 ATC Scenario Training Technology

As one of the main focuses of the FAA Academy, the Air Traffic Control (ATC) training program heavily relies on simulation-based training and is constantly looking into optimizing the use of such technologies. ATC simulation scenarios are not only used for training purposes but also are key components for conducting human performance experiments. Currently, training scenarios are generated manually from a subject-matter-expert’s (i.e. controller’s) oral or written briefing. The effort in extracting and verifying operational scenarios and translating them into a machine-understandable language is rather cumbersome and currently conducted completely manually. To address these challenges, this project developed a scenario exploration technology that provides a platform for FAA Academy trainees and instructors to exercise a variety of scenarios in the ATC domain, called ASTT. The technology provides a platform for instructors and trainees to explore various training exercises. By taking a model-driven approach and
extending the recently proposed domain-specific ASDL, ASTT provides an ATC
scenario specification and exploration platform to easily create a variety of training
scenarios.

7.1.1 Air Traffic Control Scenario Training
ATC training currently occurs in two phases: FAA Academy training and on-site
training. The Academy uses a combination of training media which includes classroom
instruction, part-task training, and technology-enhanced training. Classroom instruction is
in the form of in-class lectures which may be supplemented by the use of presentations
and handouts. Part-task training involves the use of lectures along with basic laboratory
activities (Updegrove & Jafer, 2017). Technology-enhanced training forms the majority
of the coursework by the use of low-, medium- and high-fidelity simulators for
familiarization with standard ATC tools.

The FAA Academy makes use of scenario-oriented training modules to teach trainees
how to respond to situations they encounter. While it is not possible to make such
training exhaustive as scenarios still need to be designed by instructors, it helps them get
familiar with different tools and technologies, and with the critical thinking process
which is essential in a fast-paced, high-risk environment like that experienced by
controllers.

7.1.2 Challenges of Simulation-Based Training
Given the current mission of the FAA Center of Excellence for Technical Training and
Human Performance (TTHP), ATC simulation and training scenarios are the key in
conducting human performance studies and delivering simulation part-task training. From
a recent visit to the FAA Academy, it is apparent that there is an immediate need for a
diverse pool of ATC training scenarios. Currently, training scenarios are generated manually from a controller’s oral or written briefing. This is known as the “operational scenario”, which is then verified and manually translated into the simulator’s language, providing an executable scenario script for the target simulator technology. The effort in extracting and verifying operational scenarios and translating them to a machine-understandable language is rather cumbersome and conducted completely manually.

While this process is being implemented presently at the Academy, there are other challenges to providing students sufficient simulator-based training. In addition to the availability of scenarios, a major problem facing this is the cost of the technology. Simulators in place at the Academy cost millions of dollars to obtain and maintain, and are hence used for training as much as possible. This gives students the ability to use them during class time, but they are unable to practice on their own to study at their own pace.

7.1.3 Enroute ATC Scenario Specification

En route scenarios contain several aircraft, usually a combination of active flights and departure flights, the latter of which are activated in the middle of the scenario. The enroute controller trainee’s job is to handle all traffic within their sector. This includes maintaining separation, coordinating point-outs and handoffs with neighboring sectors, issuing clearances, and responding to any issues, conflict alerts and other emergencies. For the purposes of training, several tasks are selected for the student to be tested on, and a record of their response to each situation is used for grading their proficiency.
7.1.4 ATC Scenario Exploration with ASDL

The original design of ASDL was able to model a flight from departure to landing with respect to major flight events as well as clearance deliveries. However, not much coordination or communication with Air Route Traffic Control Centers (ARTCCs) was included within the first iteration. The model had to be extended in order to explore scenarios from an ATC perspective. The extension included: (1) addition of en route terms to the ASDL ontology (ASDL Ontology, 2016), and (2) addition of model entities corresponding to newer concepts to the ASDL metamodel (Chhaya, Jafer, Coyne, Thigpen, & Durak, 2018).

The ATC terms added to the ontology were divided into three categories: Actions, Entities, and Events. Actions are responses to the Events and are performed by or through the use of one or more entities. Entities are objects or beings present in the physical world that interact with each other and are modeled within the scenario. Events are triggered proceedings that occur either with time – for instance, departures or arrivals, or when favorable conditions exist – for instance, yellow alert when the separation between any two aircraft is between five to twelve miles.

The entities, events, and actions described in the sample scenario can be seen in the ASDL Ontology, written in OWL. All these elements can be found in Figure 38. Entities are on the left and include the ARTCC (Center), Aircraft, Airport, Airspace, Controller, and Weather. Only those terms that relate directly to en route control have been defined, as entities such as an aircraft and airport have already been defined in other works on ASDL (Jafer, Chhaya, Durak, & Gerlach, 2016; Jafer, Chhaya, & Durak, 2017). Events can be seen in the middle of the figure and include arrivals, departures, holding,
IAFDOF, and various alerts. A description of each of these terms is also available in the table. Actions are shown on the right side of the figure and are performed by the controller entity. Some examples of actions are making point outs, handoff, route changes, and responding to requests.

Figure 38: Elements added to en route ontology

Once these additional elements were added to the ASDL Ontology, ATC training scenarios could be described using the tool. This process has been described for the sample scenario in the following section.

7.1.5 Scenario Specification with ASDL

ASDL can be used to define all entities, attributes, and relationships needed to specify a complete flight scenario. (Jafer, Chhaya, Durak, & Gerlach, 2016) The ASDL tool suite has been created in three stages, as can be seen in Figure 39. First, the conceptual scenario metamodel is built using Ecore in Eclipse to be used as a baseline for all scenarios. This metamodel has been built using the ATC ontology, which has been
discussed earlier and includes all the keywords and concepts that define an ATC scenario. In the second step, the metamodel is used to create an Ecore model of a specific scenario. This model does not need to be designed and implemented in Ecore directly but is rather created using the ASDL Graphical User Interface (GUI), which gives users the option to select specific scenario elements from a menu. This allows the user to work directly with the tool, without needing to understand the background code involved. At the deployment stage, the information entered within the scenario model is automatically converted into a standard scenario script in eXtensible Markup Language (XML) format. This XML script can then be utilized by end-users to execute the scenario in their target simulator.

Figure 39: Development process for ASDL scenario generator

A snippet of an XML scenario script can be seen in Figure 40. This script corresponds to the scenario described below:

The scenario has a total of 26 aircraft, and tests users on events including point outs, arrivals, departures, and conflict alerts. The scenario lasts for 45 minutes and has events occurring at random intervals in that timeframe, thus requiring the controllers to be alert and focused throughout the period.
The XML snippet shows the number of aircraft in the scenario, the airspace it is involved in, and the start and end times. Weather conditions are listed as entered, followed by any prompts and then aircraft information.

```xml
<scenario>
  <syntax_version>5</syntax_version>
  <ism_version>2.2.0.2316</ism_version>
  <name>ERAM 3</name>
  <description>26 AIRCRAFT, CBM COLD, R931 COLD, R357 COLD, VFR</description>
  <start_time>01:00:00</start_time>
  <stop_time>02:30:00</stop_time>
  <external_airspace>MasterZAE_New5J.xml</external_airspace>
  <wind>
    <wind_position>
      <id>CENTER</id>
      <position>400000N/0850000W</position>
      <wind_level>
        <altitude>0</altitude>
        <from>360</from>
        <temperature>0</temperature>
        <speed>0</speed>
      </wind_level>
    </wind_position>
  </wind>
  <prompt>
    <time>01:06:05</time>
    <acid>N5334D</acid>
    <active>1</active>
    <pending>0</pending>
  </prompt>
  <action>General Text</action>
  <text>REQUEST VISUAL APPROACH GWO</text>
</scenario>
```

Figure 40: Snippet of XML script for the sample scenario

Once the scenario begins, the aircraft are simulated as flying based on their flight data, and interact with the airspace and controller whenever an event is triggered, either based on time and pre-recorded data, or based on live environment scanning. Examples of the latter include yellow and red alerts, as well as hand-offs and point-outs when they have to be received from other sectors.

This extension makes ASDL more versatile for different facets of flight simulation and is an added step on the path towards standardization of the language.
7.2 Scenario-based Generation of Ontologies for Domain-Specific Languages

This section discusses operational scenarios that are already provided at the beginning of the modeling process and how they could be used to create a domain ontology. This approach proposes the use of scenarios in natural language, which are currently used in requirements engineering and testing, as the basis for developing an ontology for the domain model iteratively. It is another application of the DoSS idea of using natural language scenarios to learn domain knowledge, but in this case, follows the established framework of creating an ontology before proceeding with domain modeling.

There have been some published works on the use of natural-language texts for ontology learning (Shamsfard & Barforoush, 2003). Two such projects are ASIUM and Hasti. ASIUM (Faure, Nédellec, & Rouveirol, 1998) provides a methodology for learning verb frames and taxonomic knowledge, based on a statistical analysis of syntactic parsing of French texts. Hasti (Shamsfard & Barforoush, 2004) is an extensive model for ontology learning from scratch. Hasti extracts lexical and ontological knowledge from Persian (Farsi) texts. While the idea of the proposed approach is the same in the use of natural language text to build an ontology, the type and application are more specific. ASIUM and Hasti incorporate the knowledge of the entire domain through learning from extensive texts. The approach suggested here can be used to build an ontology for domain modeling using a much smaller body of input, which is the operational scenarios provided at the beginning of the modeling process (Chhaya & Jafer, 2020).

7.2.1 Methodology

This approach aims to use natural language scenarios to extract key terminology in the domain with the assumption that all key concepts would appear regularly in natural-
language scenario descriptions. This approach analyzes word frequency in operational scenarios to distinguish these important keywords. It is presumed that articles and prepositions can be ignored when looking for domain-specific keywords. Verbs can be used to determine what entities interact with each other and how within a simulation, and nouns are the chief elements that are specific to the domain. In order to obtain an accurate domain model, a large number of scenarios need to be analyzed and the evolution of the model with the addition of scenarios needs to be observed. The case study in this section will use flight training scenarios to obtain an ontology for aviation, currently specific to a flight simulation.

7.2.1.1 Single Scenario Text

First, the text of a single scenario is parsed to see if any words stand out when checking for frequency. The scenario used here has been obtained from the FAA’s example scenarios for flight instructors (Federal Aviation Administration, 2013). The first scenario chosen is:

In this scenario, the flight starts out as a VFR flight to a nearby airport. Along the way, have the student put on a view limiting device and demonstrate each of the basic attitude instrument maneuvers. At some point, put the student into an unusual attitude and demonstrate the proper recovery procedures.

Upon arrival at the destination airport, have the student demonstrate a slip to a landing. While in the traffic pattern, have the student demonstrate a go-around from a rejected landing while explaining the key elements that go into deciding when to reject a landing.
At this point, you can remain in the traffic pattern and complete another landing or depart the pattern for the return trip depending on how the student performs the approach and landings.

During the return flight, the student should have you put on a view limiting device. For safety purposes, you can simulate the role of a student wearing a view limiting device. The student should be able to properly explain each of the maneuvers to be flown as well as properly evaluate your performance on each one you demonstrate. During this time, you can help the student learn proper techniques for putting students into unusual attitudes without imposing undue stress on the airplanes or putting the flight at risk.

Word frequency analysis shows that there are 13 words in this scenario that appear at least four times in it, as shown in Table 21. After removing articles and other parts of speech, we are left with only four nouns that can be considered part of the domain: student, instructor, flight, and landing. All words in the scenario were considered in this case. Using a single scenario with 104 unique words, we have been able to distinguish four concepts relevant to pilot training and we only have to establish the relationships between them.

Table 21: Frequency of words obtained from the first scenario

<table>
<thead>
<tr>
<th>Word(s)</th>
<th>Occurrence</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>The</td>
<td>24</td>
<td>10.71%</td>
</tr>
<tr>
<td>A</td>
<td>11</td>
<td>4.90%</td>
</tr>
<tr>
<td>Student</td>
<td>9</td>
<td>4.02%</td>
</tr>
<tr>
<td>Instructor</td>
<td>7</td>
<td>3.13%</td>
</tr>
</tbody>
</table>
7.2.1.2 Addition of other scenarios from the same source

Clearly, {student, instructor, flight, landing} are important concepts in a flight training simulation, but they cannot describe a full scenario. In order to find more keywords, other scenarios have to be used. In this section, 10 more scenarios were used from the same source (Federal Aviation Administration, 2013) in order to have key concepts that are described similarly. Once these were analyzed, 439 unique words were found. An occurrence threshold of a minimum of five times in the 10 scenarios was imposed in order to consider the important domain concepts. Only nouns have been included in the model at this time and hence shown in Table 22 in order to save space.

Table 22: Frequency of nouns obtained from 10 scenarios

<table>
<thead>
<tr>
<th>Word(s)</th>
<th>Occurrence</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>89</td>
<td>3.82%</td>
</tr>
<tr>
<td>Instructor</td>
<td>73</td>
<td>3.13%</td>
</tr>
<tr>
<td>Flight</td>
<td>30</td>
<td>1.29%</td>
</tr>
<tr>
<td>Approach</td>
<td>28</td>
<td>1.20%</td>
</tr>
<tr>
<td>Airport</td>
<td>20</td>
<td>0.86%</td>
</tr>
<tr>
<td>Landing, instrument</td>
<td>14</td>
<td>0.60%</td>
</tr>
<tr>
<td>Scenario, runway</td>
<td>10</td>
<td>0.43%</td>
</tr>
<tr>
<td>Airplane, pattern</td>
<td>7</td>
<td>0.30%</td>
</tr>
</tbody>
</table>
As can be seen from the table, the domain keywords begin to take shape once additional scenarios are added. As the sample size in this case was smaller, words that are repeated up to five times were chosen; however, the threshold is much larger when there is a longer list of available scenarios.

7.2.1.3 Addition of other scenarios from different sources

Once several keywords were identified, scenarios from different sources were chosen in order to maximize the content of the domain covered. After adding 20 scenarios from Calfior and Miller (1994) to the list of 10 scenarios by the FAA (2013), the frequency of words was analyzed again to obtain the nouns shown in Table 23. This time, there were 1092 unique words, of which 120 appeared 10 times or more in the text. The occurrence threshold of a minimum of 10 times in the 20 scenarios was imposed in order to consider the important domain concepts.

Table 23: Frequency of nouns obtained from 30 scenarios from three sources

<table>
<thead>
<tr>
<th>Word(s)</th>
<th>Occurrence</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>89</td>
<td>1.29%</td>
</tr>
<tr>
<td>Approach</td>
<td>81</td>
<td>1.17%</td>
</tr>
<tr>
<td>Instructor, airport</td>
<td>73</td>
<td>1.06%</td>
</tr>
<tr>
<td>Flight, runway, departure</td>
<td>56</td>
<td>0.81%</td>
</tr>
</tbody>
</table>
7.2.2 Generating Ontology from Identified Keywords

The 30 scenarios yielded 26 unique elements for the domain. This is a very small number and can be adjusted by changing the repetition threshold in the number of occurrences or by analyzing more scenarios. However, as this is a case study, the number is considered sufficient.

7.2.2.1 Relationships between elements

Now that keywords have been identified, the domain only needs to be studied with respect to these elements in order to understand the relationships between them. Through a cursory review of the 26 keywords in aviation, the hierarchical relationships of these elements are shown in Table 24. The words in angular brackets, i.e. <text>, were not originally part of the scenarios but were discovered when studying the domain for the nouns obtained.
Table 24: Relationships between elements identified from scenarios

<table>
<thead>
<tr>
<th>Parent Element</th>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Pilot&gt;</td>
<td>Student</td>
</tr>
<tr>
<td></td>
<td>Instructor</td>
</tr>
<tr>
<td>Airport</td>
<td>Runway</td>
</tr>
<tr>
<td>Flight</td>
<td>Fuel</td>
</tr>
<tr>
<td></td>
<td>IFR</td>
</tr>
<tr>
<td></td>
<td>Route</td>
</tr>
<tr>
<td></td>
<td>Altitude</td>
</tr>
<tr>
<td>&lt;Flight State&gt;</td>
<td>Approach</td>
</tr>
<tr>
<td></td>
<td>Climb</td>
</tr>
<tr>
<td></td>
<td>Turn</td>
</tr>
<tr>
<td></td>
<td>Descent</td>
</tr>
<tr>
<td></td>
<td>Cruise</td>
</tr>
<tr>
<td></td>
<td>Enroute</td>
</tr>
<tr>
<td></td>
<td>Pattern</td>
</tr>
<tr>
<td></td>
<td>Departure</td>
</tr>
<tr>
<td></td>
<td>Landing</td>
</tr>
<tr>
<td></td>
<td>Takeoff</td>
</tr>
<tr>
<td>&lt;Air&gt; Traffic Control</td>
<td></td>
</tr>
<tr>
<td>&lt;Airspace&gt;</td>
<td>Fix</td>
</tr>
<tr>
<td></td>
<td>Point</td>
</tr>
</tbody>
</table>
7.2.2.2 **OWL Ontology based on Elements and Relationships**

The entities and relationships described above can be used to create an OWL ontology. The key parent concepts from Table 24 can be seen in the OWL format in Figure 41.

![Figure 41: High-level view of scenario-based aviation ontology](image1)

Upon expanding the high-level view from Figure 41 to see the components of the Airspace, Airport, and Flight classes, the view in Figure 42 is obtained. The Flight_State class has also been expanded in Figure 42.

![Figure 42: Components of airspace, airport, and flight class in ontology](image2)
There are very few elements described here since the threshold of repetition was set to be high and a very small set of natural-language scenarios was considered. However, the case study aims to show that the elements obtained in this manner are useful and can be used to design a domain model to create scenarios of this type. In order to understand the value of this approach, this ontology will be compared to an existing aviation ontology in the next section.

### 7.2.3 Comparison of elements with ASDL Ontology

This section discusses the previously published ASDL ontology and compares it to the version generated here. The core elements of the two ontologies are shown in Figure 43.

![Figure 43: Comparison of high-level ASDL and scenario-driven ontologies](image)

It can be seen that weather conditions did not appear often in the scenarios chosen for this approach, so the Weather class is missing. The other classes are identical, except for the addition of a superclass called “Scenario”, which is not specific to the aviation domain, and the fact that an “Aircraft” is referred to as a “Flight” in the scenario-driven version.

The “Aircraft” and “Flight” classes are compared next in Figure 44. It can be seen that the properties in the original ASDL ontology are a lot more detailed. However, the items captured in the scenario-driven ontology are able to define flights from the perspective of
a student pilot and instructor, even if they are lacking in the physical properties of the aircraft being simulated.

Figure 44: Comparison of components in ASDL and scenario-driven ontologies

It shows that a much deeper understanding of the domain was required to create the first ontology, but on the other hand, it is possible that specific flight properties and controls that have been included in the detailed ontology do not feature in the generation of a scenario for a student pilot. It is a more efficient solution when classes are only created if they are expected to be utilized instead of creating classes for every concept in the domain. It is also a safe assumption that given enough scenarios that need to be modeled, physical characteristics or flight properties would appear in a scenario-driven model as long as they are relevant to the scenario at hand. One drawback of this approach is that if
a “unique” scenario needs to be modeled, relevant characteristics might slip by if we only account for the frequency of mention.

As natural language poses its own issues, they will need to be manually solved currently, but the time saved by automatically analyzing word frequencies and finding nouns among the most frequent words instead of studying a domain thoroughly from scratch seems to be very advantageous and should be exploited.
Chapter 8

Conclusions and Future Work

This chapter discusses the conclusions from the dissertation and recommendations for future work.

8.1 Conclusions

This dissertation presented the design and development of a novel framework for domain modeling using scenario-based development techniques called the DoSS Framework. The framework uses operational scenarios that are already provided at the beginning of the modeling process to define a domain model without the need for additional domain expertise. This approach proposes the use of scenarios in natural language, which are currently used in requirements engineering and testing, as the basis for developing a domain model iteratively.

A sample domain model was created by the author and another user who is not a domain expert, in order to verify the claim that DoSS can be used without domain knowledge. The two domain models were almost identical and took a short amount of time with low scores of self-reported workload. Both models were verified and approved by domain experts, showing that the framework is a promising method of domain model development.
The dissertation also discussed real-world applications of DoSS in the creation of an aviation-specific DSL called ASDL. An Air Traffic Control training tool, ASTT, was used to understand the application of this DSL. Scenarios used in ASTT could be created using the ASTT metamodel, showing its applicability. An example was shown using the DoSS process to create an ontology-development framework using natural language scenarios.

Four research questions were posed at the beginning of the dissertation. This section aims to discuss the findings from the project in relation to the questions that were asked.

- **What is the use of standardized DSLs for scenario development in a given domain and what advantages does it offer?**

  Although the importance of scenarios in modeling and simulation has long been well known, there still exists a lack of common understanding and standardized practices in simulation scenario development. Using a DSL to provide a standard scenario specification will lead to a common mechanism for verifying and executing aviation scenarios, effective sharing of scenarios among various simulation environments, improve the consistency among different simulators and simulations, and even enable the reuse of scenario specifications (Jafer, Chhaya, Durak, & Gerlach, 2016; Jafer, Chhaya, Durak, & Gerlach, 2018).

  The publication of ASDL at the aviation M&S community had gained enough interest from both academia and industry that motivated the work towards standardizing it through SISO. The AIAA MSTC recently launched a working group towards the development of a standard simulation scenario definition language for aviation which
addresses the lack of standardized practices and common format that lead to degraded interoperability and shareability (Torens, et al., 2018). The AIAA MSTC workgroup on simulation scenario development aims at a well-formed standard language for specifying simulation scenarios. (Durak, et al., 2018). This effort for the creation of ASDL and the idea of a standardized DSL for aviation has received 20 independent citations.

- **How can the process for DSL development be refined to leverage requirements elicited from domain users without additional domain expert assistance?**

Scenario-based approaches are common for behavior modeling and testing but are not currently formalized for the design and development of models. A scenario-based approach for modeling may be preferred as it uses materials already involved in the development and testing process, but a formal ontological representation ensures that all terms are correctly defined and accounted for before the domain model is developed.

A framework for iterative development of domain models from scenarios has been proposed in this dissertation. It was used to develop a domain model, which can be compared with the model obtained using the ontology-based approach. The two domain models at this stage (for modeling a series of flight training scenarios) were shown to be very similar. However, the approach required a basic grasp of the English language and the ability to distinguish different parts of speech. While a general guideline for identifying entities and interactions was provided on the basis of language construct in English, it still requires the developer to recognize grammatical anomalies in the scenario narrative if present. The DoSS framework is a promising alternative for developing a DSL without excessive domain knowledge, but it would be more valuable if the process of identification of keywords from scenario text could be automated to some extent. The
scenario-based approach needs more clearly-defined rules and processes before it can be used to replace or be more comprehensively compared with the existing ontological approach.

- **What value does the implementation of DSLs using only elicited requirements add in terms of a) time taken and b) technical knowledge required?**

Due to the similarities in the domain models obtained using ontology-based and scenario-based approaches, the biggest factor in choosing a methodology is the amount of resources and time consumed. It was clear that fewer sources of information were required using DoSS. While the time taken was not insignificant as it required the developer to go over each scenario narrative multiple times in order to extract all the elements required from it, it was far shorter than the time taken to create a formalized ontology.

In order to understand the impact of the technical knowledge required on the domain model created, a case study was conducted with the help of a non-domain expert who tried to create a domain model using the DoSS framework with natural language scenarios. The resulting domain model was almost identical to that created by the author, showing that the DoSS framework does not rely upon technical knowledge to generate domain models.

A log of the time taken and effort required for this user also showed that the tasks were not considered tedious or taxing. It should be noted that this is only one data point and does not conclusively suggest that using the DoSS framework will always result in efficient and quickly-produced domain models which would match those created by a
domain expert, but it provides a positive response which allows for future case studies and implementation of this framework for other domains.

The final models were verified by domain experts in aviation, specifically professional pilots and air traffic controllers. DSLs are ultimately meant for domain users to perform their tasks more efficiently, so domain experts must be able to validate and verify all models. However, this dissertation found that their involvement can be minimized by using natural language scenarios so that they only need to provide scenarios and verify the final model, instead of being involved in the full domain analysis phase and without requiring the M&S expert to deeply dive into domain knowledge.

- **How can a DSL be evaluated and the process for it formalized so that it can evolve over time?**

DSLs support higher levels of abstractions than general-purpose modeling languages and are closer to the problem domain than they are to the implementation domain. Consequently, a DSL follows the domain abstractions and semantics, allowing modelers to perceive themselves as working directly with domain concepts. Furthermore, the rules of the domain can be included in the language as constraints, disallowing the specification of illegal or incorrect models (Romero, Rivera, Duran, & Vallecillo, 2007). DSM is a manner of developing systems that uses DSLs to represent the various facets of a system, in terms of models (Rivera, Durán, & Vallecillo, 2009). However, DSM approaches mainly focus on structural aspects of DSLs. The behavioral semantics of models are not explicitly defined, which can lead to a semantic mismatch between design models and the modeling languages of analysis tools. This lack of explicit behavioral
semantics strongly hampers the development of formal analysis and simulation tools (Rivera, Durán, & Vallecillo, 2009).

DoSS suggests the construction of a preliminary formal model using the scenario narrative as an indicator of changes made to domain entities in the form of statecharts. Each scenario narrative then lends itself to the addition of states and transitions over time until the full behavior of the model is encapsulated in the final statechart. The use of statecharts has previously been formalized in the SCENT Method (Ryser & Glinz, 1999) so the rationale for their use is already well-established. The DoSS framework recommends the use of one iteratively-produced statechart encapsulating the full behavior of the DSL as opposed to multiple statecharts that can be used to generate test cases as suggested in the SCENT method.

The final statechart for flight behavior discussed in Chapter 6 was verified by pilots, showing that it is possible to model this behavior form information contained within natural-language scenarios.

8.2 Recommendations for Future Work

This dissertation provided a framework for creating domain models using natural language scenarios. A preliminary case study and analysis were conducted to determine its usability which showed promising results, but this effort needs to be taken further in order to establish DoSS as a potential framework for domain model creation in a formalized setting.
8.2.1 DoSS Formalization

The DoSS framework case study showed that it can be used as an alternative method for creating domain models from natural language scenarios. However, there are still some considerations for the amount of time needed and the familiarity with language syntax and semantics in the case of scenarios. There is also a concern that the case study which led to an almost identical model between a domain expert and a non-expert’s work could be an outlier and similar experiments with other users might yield less favorable results, or at least results that are diverse enough that the framework may not account for the understanding and interpretation of the user following it.

A recommendation for future work is to combine the steps in this framework with natural language processing techniques so that entities can be separated and properly contextualized using machine learning instead of requiring manual identification of entities and elements in each scenario. This would also help with the current issue of synonyms being used in scenarios from different sources which may lead to duplicate entities being identified and having to be manually combined into a single entity.

There has been some research being done on using such techniques for ontology learning (Faure, Nédellec, & Rouveirol, 1998; Shamsfard & Barforoush, 2003; Shamsfard & Barforoush, 2004) which can be extended to domain modeling, but using only a small set of natural language scenarios so that the effort involved in creating the pool of information is lower than required presently.

8.2.2 ASDL Standardization

ASDL was created with MSDL as a source of inspiration, as both languages share a similar domain of scenario generation and application. However, ASDL addresses
scenario generation for the aviation domain, which is not addressed or supported through MSDL. Similarly to MSDL, in this research project ASDL is defined using an XML schema that allows for format standardization and content verification. This definition of ASDL will be used towards applying for SISO standardization.

The project extending ASDL for generating ASTT scenarios shows the value of the DSL for all aviation scenario generation. A proposed future effort in this direction would be to solicit other such use cases, for instance, scenario generation for unmanned aerial vehicles, to further extend the applicability of ASDL in the domain.

### 8.2.3 Ontology Development

The underlying idea of the DoSS framework was used to develop a domain ontology. An ontology is a vocabulary for a domain and describes the key concepts and their relationships. Instead of studying the domain and searching for meaningful concepts, this project approached the ontology through natural-language scenarios in the domain. The idea was that if the frequency of words in these scenarios is analyzed, key concepts become obvious as they are used repeatedly in scenarios and the ontology or domain modeling engineer only needs to investigate these specific concepts to understand the overall big picture of the domain. By knowing exactly what keywords to look for, concepts that were not mentioned in the scenarios also become easier to find.

Future work includes the generation of an aviation ontology with a much larger scope so it can be properly compared to the ASDL ontology. A large number of scenarios will need to be written or obtained for this purpose. One other concern that needs to be identified is the use of synonyms in natural-language scenarios. For instance, the terms
“flight” and “aircraft” are sometimes used interchangeably and it will be a manual process to identify the “correct” word to use when creating the ontology.

A future experiment would involve the creation of a similar ontology by a user who has no domain expertise and only a set of scenarios. The ontology can be compared to one created by a domain expert and certain metrics can be used to determine the value of this approach: time taken in both cases, the knowledge required, and the number of errors in the ontology made by the new approach. If the approach is not justified by the experiment, there may still be potential in its use to validate an ontology after its generation through other means as it would enable quick identification of missing terms.

As natural language poses its own issues, they will need to be manually solved currently, but the time saved by automatically analyzing word frequencies and finding nouns among the most frequent words instead of studying a domain thoroughly from scratch seems to be very advantageous and should be exploited.
References


Federal Aviation Administration.


Taba, N. H., & Ow, S. H. (2012). A Scenario-Based Model to Improve the Quality of Software Inspection Process. *2012 Fourth International Conference on Computational Intelligence, Modelling and Simulation* (pp. 194-198). IEEE.


