Generic Navigation of Concerns and Perspectives in TouchCORE

Ian Gauthier

Department of Electrical & Computer Engineering

McGill University

Montréal, Québec, Canada

February 15, 2021

A thesis presented for the degree of Master of Engineering

Abstract

The practice of Model Driven Engineering (MDE) recommends the creation of several different models which describe different aspects of a product in order to create a holistic view of the proposed system. In order for a tool to properly facilitate the development of software models, it must allow a user to properly navigate between these different models and their elements. Due to the ever growing number of possible languages in which a system aspect may be expressed with a model, an approach to navigation which is language independent can provide many benefits to a user such as a more declarative way of specifying navigation for new languages instead of hard-coding navigation for each new language.

In this thesis, we outline our implementation of a generic navigation system within the software modeling tool TouchCORE. We utilize the concept of a Perspective which is used to combine several languages and map elements from different models which represent the same concept to each other. Based on a proposed navigation metamodel, we introduce a system within TouchCORE for defining the pertinent concepts of a given language and

pertinent navigation mappings within a language or between languages within a Perspective. We then implement a navigation bar used to display these important elements to the user when creating or editing a model. This navigation bar also provides the user with a series of navigable links to other models or concepts within a Perspective. Finally, we introduce a test suite used to validate the quality of our implementation.

Abrégé

La pratique de l'ingénierie dirigée par modèles (IDM) recommande la création de plusieurs modèles différents qui décrivent les particularités d'un produit afin de créer une vue globale du système proposé. Pour qu'un outil facilite effectivement le développement des modèles logiciels, il doit permettre à l'utilisateur de naviguer facilement à travers des différents modèles et leurs éléments associés. En raison du nombre croissant de langues possibles dans lesquelles un aspect du système peut être exprimé en utilisant un modèle, une approche qui favorise une navigation plus indépendante de ces langues peut offrir de nombreux avantages à l'utilisateur, notamment sa manière de spécifier la navigation d'une forme plus facilement interprétable dans de nouvelles langues au lieu de coder en dur la navigation pour chaque nouvelle langue.

Dans cette thèse, nous décrivons notre mise en œuvre d'un système de navigation générique au sein de du logiciel de modélisation TouchCORE. Nous utilisons le concept de 'Perspective' qui permet de combiner plusieurs langues et de cartographier les éléments de différents modèles qui représentent les mêmes concepts. Sur la base du métamodèle de

navigation proposé, nous introduisons un système dans TouchCORE qui permettra de définir les concepts pertinents d'une langue donnée et des cartographies de navigation appropriés dans une langue ou entre les langues dans une Perspective. Nous mettons ensuite en place une barre de navigation utilisée pour afficher ces éléments importants à l'utilisateur lors de la création ou de la modification d'un modèle. Cette barre de navigation fournit également à l'utilisateur une série de liens navigables vers d'autres modèles ou concepts au sein d'une Perspective. Enfin, nous introduisons une suite de tests utilisés pour valider la qualité de notre implémentation.

Acknowledgements

I would like to express my gratitude to my supervisor Gunter Mussbacher for taking me on in his lab and for his constant help and guidance throughout the process of crafting and writing this thesis. Without his help I would not have been able to complete this work on time. In the same vein I would like to thank Jörg Kienzle for effectively taking me on as a student as well over the past year. Finally, I would like to thank all the members of the TouchCORE team, particularly Hyacinth Ali with whom I worked closely during the course of creating this thesis, for always being available to help when I had questions with the coding work of this thesis.

I would like to thank my parents for their love and their kindness - I have never needed to worry about having your support and for that I am endlessly grateful. Finally I would like to thank the many friends who have encouraged and checked in on me over this past weird year - particularly my roommates who have kept me sane and in good spirits throughout. I appreciate you all beyond words.

Contents

List of Figures

List of Tables

List of Acronyms

CORE Concern-Oriented Reuse.

CRUD Create, Read, Update, Delete.

DSML Domain Specific Modeling Language.

EMF Eclipse Modeling Framework.

GUI Graphical User Interface.

HCI Human-Computer Interaction.

RAM Reusable Aspect Model.

SPL Software Product Line.

UML Unified Modeling Language.

Chapter 1

Introduction

The practice of Model Driven Engineering (MDE) [3] recommends the use of several different models which can be used in tandem to describe the overall construction of a given software system. When programming within this methodology, models representing different facets of the system often overlap resulting in elements being present in multiple different models of the same system. These different models often draw from multiple modeling languages and/or contain several different views within the same model in order to achieve the goal of completely describing the system.

When describing a system with multiple models, it is important for a user to be able to easily move between different models (which are often related) and different parts of an individual model in order to easily view the current description of the system - and understand it more fully - and to update the contents of the models as needed. Several

popular UML tools utilize navigation systems for this purpose, such as StarUML [4] and MagicDraw [5] which both contain tools to allow users to move between models of different languages in a streamlined way. In addition, due in part to the increased acceptance of Domain Specific Modeling Languages (DSML) [6], it is not sufficient for the navigation of models to be limited to a small set of predefined languages. Rather, it is necessary for the navigation to be amenable to a wide range of different languages and be easily extended to support new languages as they inevitably are introduced.

1.1 Motivation

In the context of this thesis, we consider the navigation system of TouchCORE [7] - a software modeling tool currently home to an expanding variety of languages. TouchCORE is a modeling tool built to facilitate both the description and use of different modeling languages [8] [1] as well as the reuse of software concepts [9] through the use of Software Product Lines (SPL) [10].

TouchCORE has grown to include the definition of Perspectives [8] [1] - a concept implemented to allow users to seamlessly describe a software system using several different languages while ensuring consistency between concepts which span multiple models. Perspectives have greatly increased the available combinations of different models within TouchCORE and create the possibility of many more being added in the near future. The current navigation system is hardcoded into the system for each language that is currently

present. As a result, when a new language is added to the system, members of the TouchCORE team must specifically implement the intended functionality of the navigation system largely from scratch and integrate it into the existing tool. This process requires a considerable amount of work from the developer any time they want to add a new language to TouchCORE.

As such, TouchCORE now requires a generic approach to navigation which can be easily expanded as new languages and combinations are introduced to the system. In addition, each of these models and Perspectives interacts with the Software Product Line models that represent the basis of TouchCORE. As navigation represents an important component of successful development, a tool that can perform these functions is of great importance to TouchCORE as its capabilities expand.

1.2 Contributions

In this thesis, we outline the functionality and implementation details of a newly integrated generic navigation system within TouchCORE. Specifically, we focus on the incorporation of a navigation tool within the context of the Perspective system which is currently being introduced into TouchCORE. The specific contributions to the navigation system are as follows:

- A generic system for navigation within an individual model which allows users to easily

determine what concepts exist within a given model, highlight specific elements and which updates dynamically to respond to changes within the model.

- A generic system to facilitate navigation between different models within a Perspective and between elements located in different models which are mapped together through the Perspective system.

- Navigation within multi-view models both between views and within a single view of the model.

- Support for Software Product Line models - specifically Feature Models [11] and Goal Models [12] which are the basis for the concern - the basic building block of development within TouchCORE.

- Support for navigation of reused models and concerns.

- A system for defining the specifics of navigation for modeling languages which are newly implemented into the TouchCORE tool and for navigating newly defined Perspectives.

Along with the description of these contributions, we outline the testing used to verify that each of the aforementioned systems are able to conform to the needs of TouchCORE users.

1.3 Thesis Outline

We now outline the thesis:

- Chapter 2 - Background: In this chapter we introduce the concepts required to understand the contributions outlined in the rest of the thesis. We specifically introduce the concepts of Domain Specific Modeling Languages, Perspectives, the TouchCORE tool, Aspects, and RAM models.

- Chapter 3 - Overview of Generic Navigation: In this chapter, we outline the specifics of our intended navigation tool and justify the need for the features included in the system.

- Chapter 4 - Implementation: In this chapter, we first outline the metamodel of the navigation system as well as describe its evolution through development. Later, we outline the specifics of the implementation of the navigation system.

- Chapter 5 - Testing: This chapter centers around the tests run to validate the navigation system and outlines our testing methodology.

- Chapter 6 - Related Work: In this chapter, we perform a survey of existing software modeling tools and compare their navigation systems with the one outlined in this thesis.

- Chapter 7 - Conclusion: In this chapter we summarize our contributions and discuss opportunities for future work.

Chapter 2

Background

In this chapter, we define the concepts necessary for understanding the context in which this thesis is written. This includes the definition of several terms relevant to the development of the TouchCORE tool and models which can be described within the tool.

2.1 Domain Specific Modeling Languages

When trying to create a software based product within a given field, one large area of difficulty can be the disconnect from experts in the field who possess knowledge of the specifics of how the system should function and programmers who are implementing the product and whose knowledge is based largely in the programming languages which will be used to encode it. In order to create a successful product in a specific domain, it is necessary to remedy this disconnect so that the required functions of the system can be specified by the

domain experts and then encoded by the programmers in a way that conforms to the needs of users. However, attempting to fix this problem verbally or without the aid of the correct systems can require considerable time and resources - leading to the need for a technological solution to this problem.

The solution to this gap can be provided by a Domain Specific Modeling Language (DSML) [6]. These are modeling languages which include domain specific concepts - allowing for the structure of the system to be outlined graphically by the domain experts without requiring an understanding of traditional programming languages which have a comparatively lower level of abstraction. These languages can also be created with the ability to generate code which represent the high level definition of the system which is as yet to be fully implemented. This allows for the domain experts to use a DSML to clearly define the outline of the system without the need for specific programming knowledge before handing off the project to developers who can build upon the structure to complete the implementation of the product.

2.2 Perspectives

In many cases, it is not sufficient to use a single modeling language to describe the entirety of a system. As such, models from multiple different languages are often used to describe a system - with each having the goal of outlining a different facet of the overall construction - as well as prescribe the actions available within the system. Due to the fact that these

individual models are used to outline the same product, it is likely that they will overlap each other. As such, it is important that the implementations of the various models are kept consistent to ensure that the final set of models results in a coherent whole. Users tasked with the goal of ensuring this consistency are likely to find difficulty if not provided help in doing so.

While there are several proposed solutions to this problem, the one that is largely considered in this thesis is the use of Perspectives [8] [1] to ensure consistency between interconnected models. A Perspective is a grouping of different modeling languages - or a single modeling language being used in multiple different roles - which, taken together, are able to represent the whole of a system better than a single language is able to. In addition, Perspectives are constructed to aid users in ensuring that when modeling a system the set of models remains consistent throughout by either preventing the creation of inconsistencies or by highlighting and removing inconsistencies after they have been introduced.

While Perspectives can be created for a single modeling language in a specific role, this thesis largely focuses on how they are used to model multiple different languages or a single language which is used in different roles. A role, in this context, is used to describe the specific function that a given language is performing. For example, in certain situations, the Class Diagram language may be used to perform the role of a Domain Model while in others it can be utilized as a Design Model. Due to the similarities between these two

types of models, the Class Diagram language can perform the function of both with only slight changes in functionality (e.g., operations are allowed only in Design Models but not in a Domain Model). As such, the Design Model and Domain Model are referred to as two different roles of the Class Diagram language. In a Perspective, both of these roles may be present and, in these cases, are handled as independent languages. Each of the languages which are used in a Perspective is assumed to have no prior connections or constraints with the other languages in the Perspective and is defined by the specific language actions which are available to a modeler.

Language actions allow for the creation and editing of models within a language and, in general, allow a user to add or remove elements from the model, change the qualities of an element or connect elements within the model. An example of a language action could be creating a class within a Class Diagram or removing an actor from a Use Case Diagram. Language actions are at a higher level of abstraction than CRUD (Create, Read, Update, Delete) operations and one language action may consist of many CRUD operations. While the languages themselves - and by extension their language actions - are independent before being added to a Perspective, the corresponding Perspective actions may either be independent of other languages or interconnected, causing a change in another model when they are performed to foster consistency between models. A Perspective action which only affects the current model is referred to as a re-expose action because it reuses a language action as is. A Perspective action which changes a language action to affect other models

within the Perspective is called a redefine action. As the former acts in a similar manner to how they would act outside of a Perspective, we do not go into more detail on them here.

To illustrate the benefits of using a Perspective to define interconnected models, we turn to an example based on a Perspective created from two languages - the Class Diagram and Sequence Diagram languages. Many actions taken within each of the individual models, such as the creation of a class within the Class Diagram, do not require any changes within the other models within the system - as such these are re-expose actions and function as they would outside of the Perspective. However, in certain cases, changes in one model must propagate to other models. Sequence Diagrams are used, in this context, to define the behaviour of a single operation within the Class Diagram. As such, if the name of the operation were to change within the Class Diagram, the name of the Sequence Diagram must also change to remain consistent with the operation and vice versa. Thus, a redefine action is created to ensure that whenever the operation name or Sequence Diagram name is updated by the user, the other is updated accordingly. As such, through explicit definition of the actions within a Perspective, a user can be more confident that the models they create will remain consistent as they edit them.

Within a Perspective, Language Element Mappings are used to explicitly define how elements from the metamodels of different modeling languages are interconnected. However, these connections are not directly visualized when a user is modeling a system within a given Perspective. This may lead to difficulty for modelers when trying to understand how the

elements of a given model are interconnected with elements of another model at run-time. Much of the work in this thesis is based upon ensuring that users are quickly able to see how changes made to a specific model affect others within the system and allowing users to quickly move between models within a Perspective.

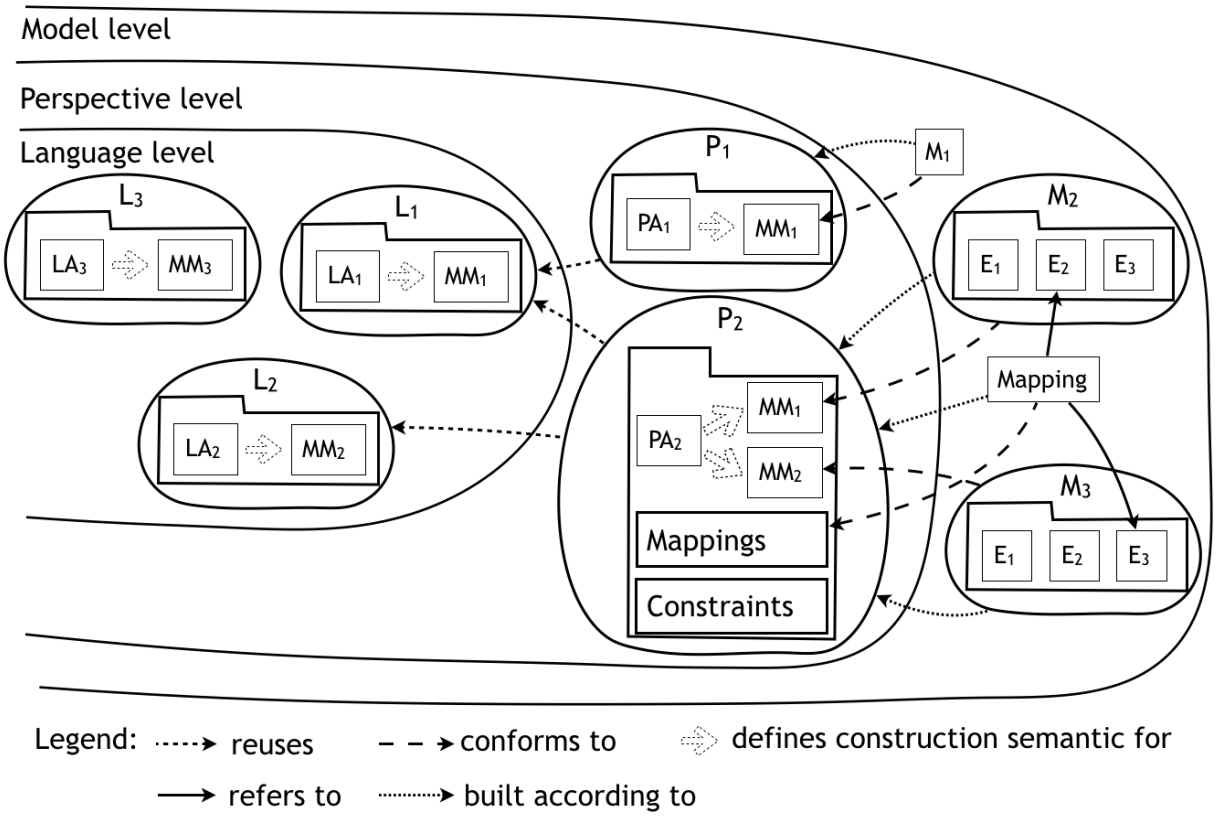


Figure 2.1: A generic description of the architecture of Perspectives along with related Languages and Models associated with the Perspectives. Initially created by Ali *et. al.* [1].

Figure 2.1 describes a generic version of the three levels pertinent to the creation of Perspectives. The Language level holds three languages (L) which have been implemented by language designers. Each of these languages contains a metamodel (MM) describing the

language concepts as well as a set of language actions (LA). The Perspective level details a set of two Perspectives which draw on one or more of the languages and reuse their metamodel(s). The Perspectives also draw on the Language actions of their relevant language(s) in order to create a set of Perspective actions (PA) either by re-exposing or redefining them. P1 represents a single language Perspective and, as such, each of the Perspective actions simply re-exposes one of the language actions from L1. P2 represents a multi-language Perspective and thus draws on both L1 and L2 to create a set of Perspective actions through re-exposing and redefining. P2 also contains a list of Language Element Mappings to define how elements from L1 and L2 are interconnected. Within the Model level are a set of models created from one of the defined Perspectives. M1 represents a model which has been created from the single language Perspective P1 and thus does not need to maintain any consistency with other models. M2 and M3 are a pair of interconnected models which are defined based on P2. They each have a set of elements which draw on one of the metamodels present within P2 - and by extension drawn from L1 and L2. In addition, they utilize the mappings defined in P2 to ensure consistency of interconnected model elements.

To exemplify the purpose of each of the three levels in Figure 2.1, we turn back to the case of the inter-related Class Diagram and Sequence Diagrams. In this case, the Language level would house information related to the Class Diagram (L1) and Sequence Diagram (L2) languages including their respective metamodels (MM1, MM2) and their lists of language actions (LA1, LA2). Within the Perspective layer, a multi-language Perspective

(P2) would be used to define the relationship between the two languages. The Perspective would contain both of the metamodels (MM1, MM2) as well as the set of Perspective actions (PA) such as those outlined above. Finally, when initializing models, instances of both Class Diagram (M2) and Sequence Diagram (M3) will be created within the Model layer along with mappings between pertinent elements - such as between operation names within a Class Diagram and Sequence Diagram names - which will ensure that these model instances remain consistent to the mapping rules set within the Perspective layer.

2.3 TouchCORE

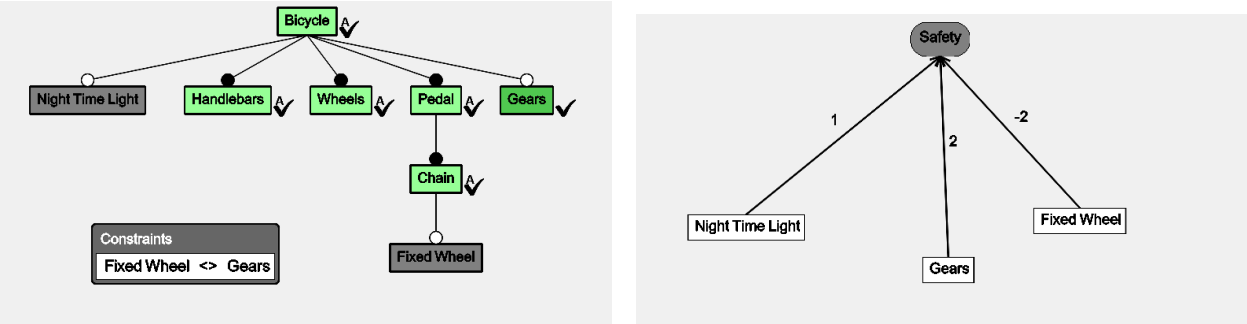
Software reuse is the process creating a new piece of software by building on previous work rather than starting work from square one [13]. Concern-Oriented Reuse is a framework that has been developed to facilitate this method of development [9]. The framework is based around the concept of a Concern which is a specified unit of reuse which allows for the development of new products. We now speak briefly on the structure of a Concern as the main developmental work of this project has been done in TouchCORE [7], which is a product that is built to facilitate Concern-Oriented Reuse and whose main unit of development is the Concern. However, we do not delve into specific detail regarding the methodology of Concern-Oriented Reuse as it is not the focus of this thesis.

A Concern can be thought of as an instance of a Software Product Line (SPL) [10] which is a grouping of software systems which share a common set of features and which are

implemented from a common set of artifacts. A set of programs within SPL development is referred to as a family of systems and when a new system is developed within the family, the common properties of the family are explored first before delving into the specifics of the new system. Within a family, a feature represents a piece of functionality which may or may not be present in a specific product. Within Software Product Line development, features are often grouped into a feature model [11] in order to specify the structure of the SPL and determine the relationships between different features.

A common form of feature model is that of a feature diagram - an example of which can be seen below for the simple situation of modeling a bicycle in Figure 2.2a. A feature diagram models the various features of a system in a tree structure with certain features acting as the parents of other features. Features may be mandatory or optional based on the status of their parent feature and also can require that other features be present or exclude other features within the diagram. Within a Concern, the feature diagram acts as the launching point for further development and, as such, is the first model shown when opening the Concern.

The second type of model found within a Concern is the goal model [14]. Goal models are often used early on in the development of a product to facilitate the definition of the requirements of a system [15]. They allow users to analyze high level objectives of a system and see how different approaches affect the completion of said objectives. Within a Concern, the standard method of displaying goal models is through the use of an impact model - an



(a) A simple feature model modeling a bicycle. **(b)** A simple goal model made in TouchCORE

Figure 2.2: An example of a simple feature model and goal model created in TouchCORE.

example of which can be found in Figure 2.2b. A goal is displayed along with features or other goals which may contribute to the completion of the main goal of the model. The features impart contributions on the goal to visualize whether they have a positive or negative impact on the main objective of the model. Higher positive numbers represent features which have a larger positive impact on the goal and negative values point to features having a negative impact on the goal. The features found within the impact model(s) are a subset of the features defined within the feature diagram of the Concern.

In addition to these two types of models, Concerns are also populated with a set of realization models which may come from a variety of languages. These realization models are related to individual features found within the feature model and are tasked with describing a specific implementation of those features. Together with the feature and impact models, the realization models round out the definition of a Concern.

As mentioned briefly above, TouchCORE (CORE here standing for Concern-Oriented

Reuse) is a tool built in order to aid in Concern-oriented development [9]. Within TouchCORE, the Concern is treated as the basic building block upon which to define a software system. When developing within the tool, a user is initially shown the Concern for their chosen product with other languages functioning as realization models of individual features defined within the feature model for the Concern. The development and testing work for this thesis has been performed within the TouchCORE tool as has the implementation of Perspectives as defined in 2.2. Thus, from this point on, all development can be assumed to be performed within TouchCORE unless otherwise noted.

2.4 Aspects and RAM

TouchCORE has been built as an improvement on the aspect-oriented modeling software TouchRAM [16]. As such, much of the functionality of TouchRAM is still pertinent to the development of this thesis - specifically the RAM (Reusable Aspect Model) [17] language which is used as an important test case for the navigation system in Section 3.2.1. As such, we briefly outline the structure of an Aspect Model now.

Aspects in TouchRAM - and therefore TouchCORE - are a multi-view model which is used to model several facets of a system within a single modeling language. Often, as noted in Section 2.2, a system description requires modeling in multiple different ways. Whereas in a Perspective this is done through connecting disparate languages - or the same language used in different contexts - a multi-view model collects several different representations of the

system into a single modeling language with several interconnected views. We now overview the specific case of a RAM aspect and its multiple different views.

The first view seen when opening an Aspect is the Structural View which functions as a UML [18] Class Diagram. Classes are used to describe the overall structure of the system and can include operations, attributes and a class hierarchy in much the same way as a normal Class Diagram. Each Aspect contains a single structural view. In addition, each class within the structural view has a State View associated with it. Much like how the Structural View is a Class Diagram representing the system, the State View is represented by a UML state diagram which defines the protocol of the class. Finally, the Aspect contains several Message Views which are associated with operations within the Structural View. Message Views are represented by UML Sequence Diagrams and describe the operation and how it functions when executed. As is clear, each of the different views within an Aspect are interrelated - with the whole of the model describing more of the system than would be possible with any one model.

2.5 Summary

In this chapter, we introduce several concepts which are necessary for the understanding of the contributions of this thesis. Specifically, we outline the use of Domain Specific Modeling Languages, the concept of a Perspective, the TouchCORE tool, and the use of Aspects and RAM models. In the next chapter, we outline the intended functionality of a generic

navigation system within TouchCORE.

Chapter 3

An Overview of the Aims of a Generic Navigation System

Within a modeling tool such as TouchCORE, it is necessary to be able to navigate between models and their elements to allow users to easily view different facets of their work. To this end, a system needs to be implemented to provide users with this functionality in a streamlined way. Due to the increased breadth of available modeling languages, we aim for this functionality to not only be available to users under a predefined set of languages. Rather, we seek to create a system which is able to handle any language which a user defines within the tool. In addition, we intend for our system to be able to navigate between different modeling languages which are defined within a Perspective (see Section 2.2). In this chapter, we outline the specifics of our goals for the system as well as motivate the specific needs that

shaped them.

3.1 Intra-Model Navigation

The most basic form of navigation that should be available is movement between concepts found within the same model. When modeling a large system with many parts, it can become difficult for a user to keep track of the position of all facets of a system at once. As such, streamlined navigation between these concepts is necessary to provide a comfortable user experience. Inherently, this type of navigation consists of only concepts of the same language (henceforth referred to as Intra-Language navigation).

Figure 3.1(a) shows a fairly straightforward Class Diagram that a user might create to model the concepts of an airline booking system. Even within a simple system such as this, which does not contain the full functionality which would be required for a comprehensive modeling of an airline booking system, the amount of elements present quickly balloons to a number which may be difficult for a user to keep track of. As such, a navigation system is tasked with accounting for all of the elements which are present within the model as well as presenting them to the user in a way that allows the modeller to quickly discern this information.

To solve this issue, we propose the use of a navigation bar which can contain a list of the elements present in the given model. An example of the proposed functionality is shown in Figure 3.2a which lists all of the pertinent model elements from the corresponding Class

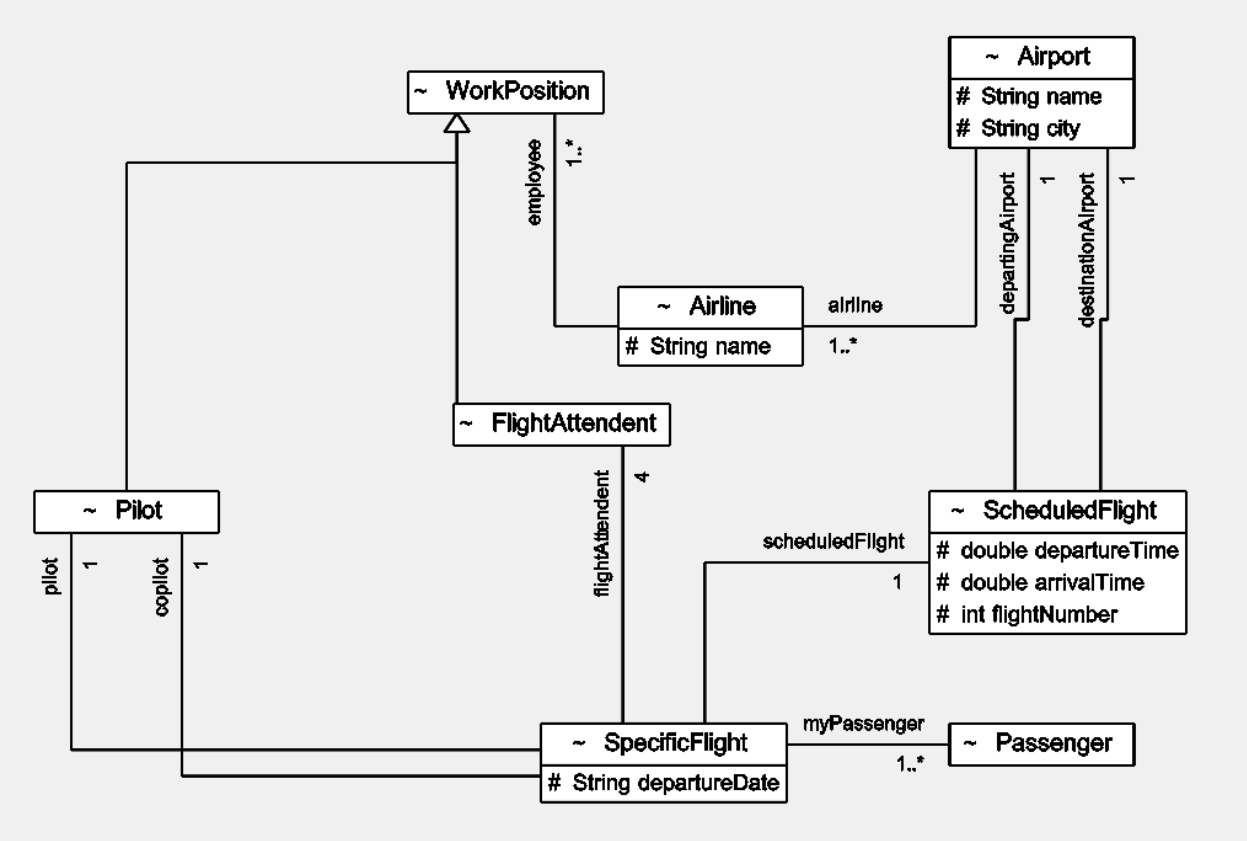
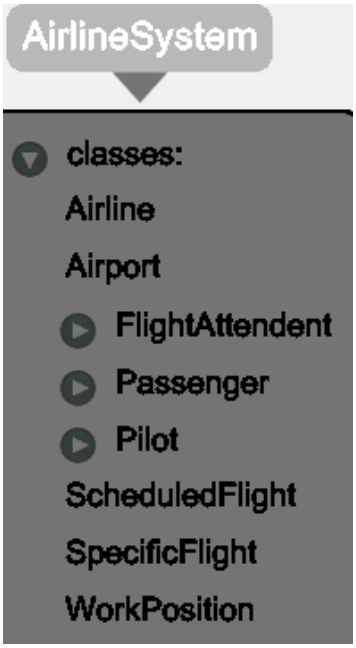


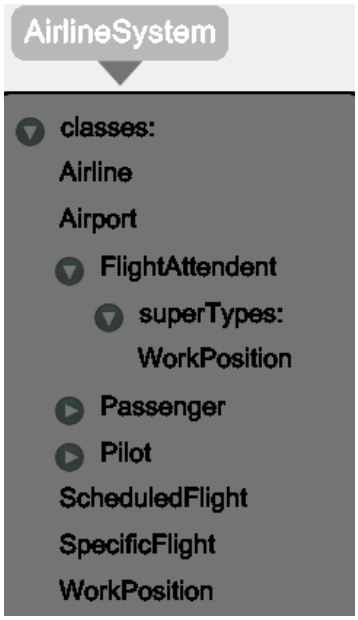
Figure 3.1: A Class Diagram showing some of the concepts that would be used to model an airline booking system.

Diagram. In this implementation, all of the classes from within the system are shown in alphabetical order so that a modeller may easily find any concept from within the system or discern that the element is not present.

The goal of a navigation system being utilized within a single model is to allow users to orient themselves within the model. To streamline this process, when a user clicks on one of the elements that belongs to the current model in the navigation bar, it should highlight the element in the model in order to clearly define where it is located. This functionality



(a) The navigation bar with only the base information shown.



(b) The navigation bar with the class hierarchy shown for one class.

Figure 3.2: Two views of the navigation bar for the Class Diagram in Figure 3.1.

can be seen in Figure 3.3 which displays the same Class Diagram found in Figure 3.1 after the WorkPosition class has been selected within the navigation bar.

In addition to displaying some of the model elements within a system in a concise manner, it is also important to present a visualization of how the different parts of the model are interlinked. For instance, in a Class Diagram, it may be useful for a user to be able to view the hierarchy of a given class so that they may be able to understand which classes extend other classes in the model. As such, the navigation system should also be able to recognize links between given model elements. To display this clearly, the navigation bar should display the super-classes for a given class underneath the listing of a given class. This

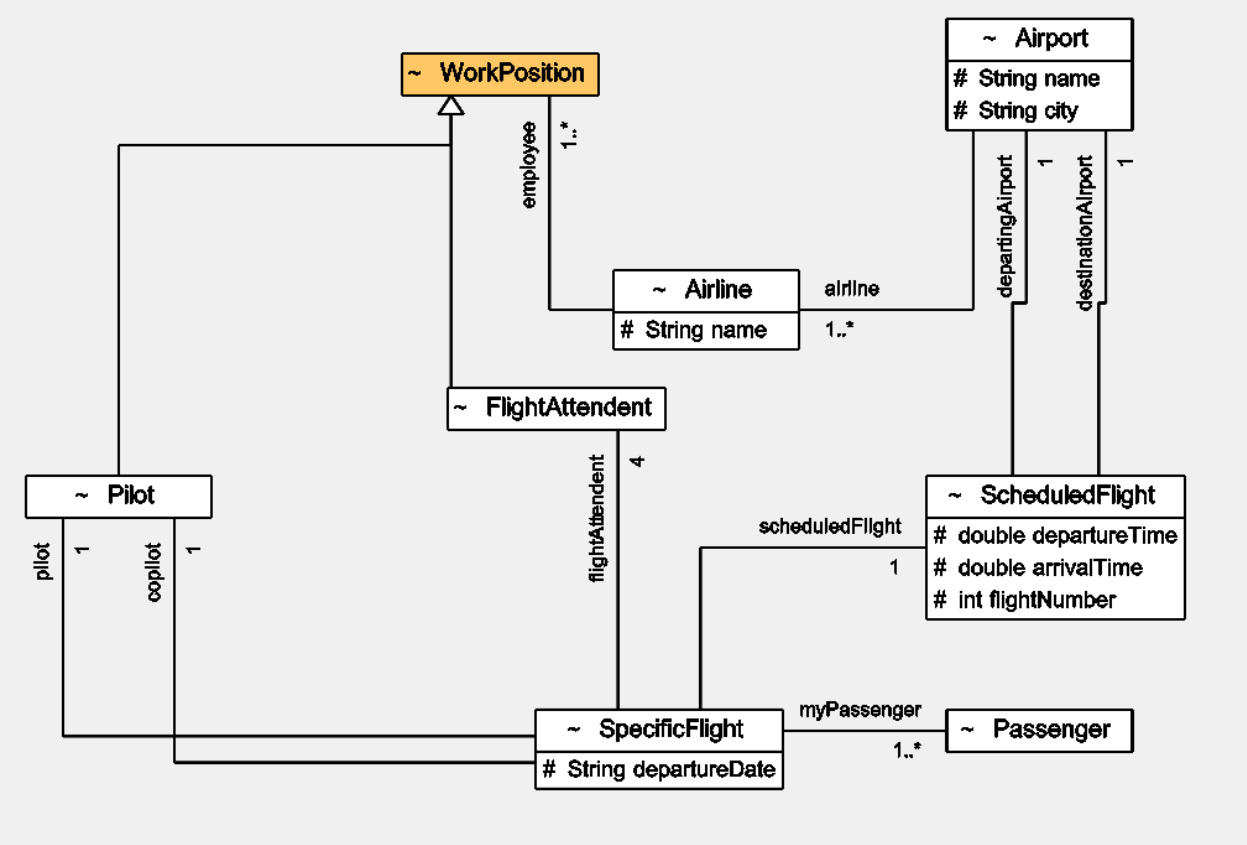


Figure 3.3: The Class Diagram shown in Figure 3.1 after the WorkPosition class has been selected within the navigation bar.

type of connection between different model elements within a single model is referred to as an Intra-Model (and thereby Intra-Language) mapping. Mappings of this kind provide the basis of generic navigation within a single model.

When viewing Intra-Model mappings such as this, a user should be able to toggle the mapping on and off depending on whether the information provided by the mapping is pertinent at a given moment. This is achieved by presenting the user with a drop down menu for elements which have mappings associated with them. In the case that a user

wishes to view the links related to a specific element, they may open the menu to view other elements which are mapped to it. As an example, in the Class Diagram, if a user wishes to view the class hierarchy of the Pilot class, they may toggle open the drop down menu to the left of Pilot in the navigation bar. This opens up the drop down menu as shown in Figure 3.2b to reveal the hierarchy information. When they finish viewing this part of the system, the modeller can simply close the drop down menu so that the screen is no longer cluttered with information on the Pilot class.

Not all of the information present within a model is valuable for navigation. For instance, in the case of the Class Diagram, it may not be very common for a modeler to navigate between different parts of the model based on a specific attribute of a class. In addition, given that navigation based on attributes is rare, including them within the navigation bar may represent a hindrance to the user as they may make it difficult for the user to find the parts of the model which are of interest. In cases such as this, it is important that only parts of the model which are useful for navigation be included within the navigation bar. However, it would not be useful to a modeler to have to make these decisions during every instance in which they wanted to create a new model. Thus, the concepts which are to be shown in the navigation bar should be defined at the system level and be automatically generated for each new model.

To facilitate this automatic generation, we turn back to the concept of an Intra-Language mapping. In order to ensure that only pertinent concepts are shown, only model elements

which are part of an Intra-Language mapping are included within the navigation bar. When the user generates a given model, the mappings between concepts within the model should also be generated and define which parts of the model are valuable for navigation and which of them should not be included. In Section 4.2.1 we outline the method by which language designers can define the Intra-Language mappings for an individual language and thereby what types of elements are included in the navigation bar for an instance of that language.

Thus far, we have only outlined the way that the navigation bar should function for a static model where all of the elements have already been defined. However, this does not accurately represent the way in which a user is likely to use a product such as TouchCORE. Indeed, when attempting to model a system, users are likely to wish to add or remove elements and connections found within their models dynamically. When making these changes, they are likely to still wish to see the state of the system within the navigation bar and move around the model to view the pertinent portions. As such, the navigation bar should be able to respond to changes in the model as they occur and allow a modeler to immediately see how changes made to the model affect its overall structure.

When a user adds an element to the model, the system should automatically check whether that element should be included in the navigation bar - based on the specifications outlined above. In the case that the new element is of a type that should be seen within the navigation bar, it should be added to the correct section of the bar. In addition, any connections that it has should also be displayed within the section. In the inverse case, when

an element is removed from the model, it should also be removed from the navigation bar. When this process occurs, all of that element's connections to other model elements should also be removed at the same time. This ensures that a user does not try to navigate to a part of the model that no longer exists. Finally, if a user changes the name of the element, that change should be propagated to all instances found within the navigation bar. This functionality should ensure that the navigation bar accurately reflects the current state of the model at all times.

3.1.1 Requirements

We now formally list the requirements for Intra-Model navigation within TouchCORE:

3.1.1: The navigation system shall be able to outline what concepts exist within a given model while also being concise enough to not overwhelm the user with information.

3.1.2: The navigation system shall make clear how elements of a model are related to each other within the structure of the navigation bar section for that model.

3.1.3: The navigation system shall automatically update the navigation bar section of a model to reflect changes made to the model such as adding elements, deleting elements, or changing their contents.

3.1.4: The navigation system shall allow the user to locate a model element within the current model.

3.1.5: The navigation system shall automatically generate navigation bar sections to reflect mappings defined by language designers.

3.2 Inter-Model Navigation

As describing a software system often necessitates the creation of several interlinked models, navigation within a modeling software cannot be confined to only movement when navigating within a single model. Ideally, a modeler is able to navigate between the different models describing a system with the same fluidity that is afforded to them within a single model. To facilitate this, we must extend the functionality described in Section 3.1 beyond the confines of a single model. In addition, given that different models describing the same system often represent disparate modeling languages, we must also account for navigation between models of different languages. Navigation of this kind is referred to as Inter-Language navigation.

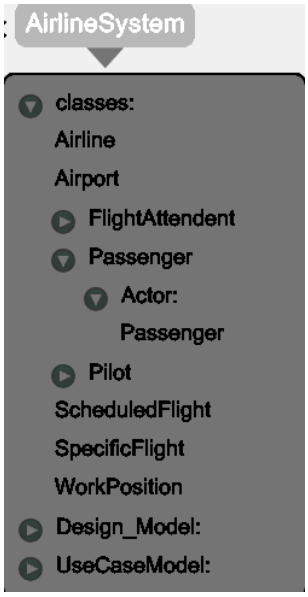
Within TouchCORE, navigation between languages is generally performed using the functionality of a Perspective. A Perspective, as described in Section 2.2, is an interlinking of several different languages (or instances of the same language being used in different roles) where language concepts are mapped to each other to facilitate the creation of multiple connected models. An example of this would be an extension to the Class

Diagram shown in Figure 3.1 where a user may wish to describe a Use Case Model for the airline booking system alongside the Class Diagram definition above to gain a more holistic understanding of the system. As these two models are inherently connected, a user is likely to want the option to quickly navigate between the two models. As such, the navigation bar should support this type of navigation so that the user can, with a single click, switch to another model describing the system.

Figure 3.4a shows the proposed setup of the navigation bar in the case of interlinked models from within the same perspective. These types of links are referred to as Inter-Model mappings and, in the case that the two models are of different modeling languages, are more specifically referred to as Inter-Language mappings. In a similar manner to the classes shown in Figure 3.1, a drop down menu is available showing each of the other models which should be navigable from the current model. The name of the drop down menu should be based on the role of the model within the given perspective. Once that drop down menu is opened, clicking on the model within the navigation bar should navigate to the view of that model (multi-view models are outlined briefly in Section 2.4) so that it is now the model currently on screen.

In addition to navigation directly between models as described above, modelers may wish to navigate between related concepts in different models. For instance, in the case of the connected Class Diagram and Use Case Diagram, the actors described in the Use Case Diagram view of the system may be related to the classes defined within the Class Diagram.

(a) The navigation bar with the Inter-Language mappings to other models shown. The Domain Model is mapped to a Use Case Model and a Design Model.



(b) The navigation bar with Inter-Language mappings between elements shown. The class Passenger is mapped to the Actor Passenger in the Use Case Diagram.

Figure 3.4: Two views of the navigation bar for the Class Diagram in Figure 3.1 with the Inter-Language mappings included.

As such, a user may be looking at a specific class and wish to see how the actor related to that class is positioned within its respective diagram. To allow for the user to understand these connections more easily, the navigation bar should list the links between model elements of the current model and their related model elements in connected models in the same way that it lists the links between the models themselves. Therefore, in addition to showing the Intra-Language mappings of a given class, the drop down menu for that class should list the Inter-Language mappings associated with the given class. These Inter-Language mappings

should be classified alongside their Intra-Language counterparts as the user is likely not to see a meaningful difference between navigation between elements within the current model and navigation to an element in another model. An example of this functionality for the Class Diagram found in Figure 3.1 can be seen in Figure 3.4b.

When a user selects one of the model elements in an Inter-Language mapping defined within the navigation bar, the system should again perform a view switch to the model containing the selected model element. It should also, similar to the functionality for selecting an element within the same scene, highlight the specific model element within the newly displayed model - allowing the user to see where in the model it is located.

In either instance of navigating an Inter-Language mapping, the navigation bar should update to reflect the new model that is currently being displayed. A new section should be added to the navigation bar which describes all of the pertinent concepts within the model as outlined in Section 3.1 as well as all of the Inter-Language mappings as described above. However, given that modelers may make frequent switches between the models describing a single system, simply adding a section to the navigation bar when this switch occurs may result in a bloated navigation bar which no longer provides use to the user. As such, when navigating an Inter-Language mapping, the navigation bar element for the original view should be removed from the bar.

3.2.1 Single Language Multi-View Navigation

Thus far, our outline of navigation has been limited to either navigation within a specific model or navigation between models of different languages. However, this does not encapsulate the full range of possible models within TouchCORE. Specifically, this dichotomy leaves out the possibility of a model which contains multiple different interlinked representations.

An example of this situation can be found within the RAM language outlined in Section 2.4. In this language, several types of representations of the same system are collected into a single language. This idea is similar to the interrelated models which were used to describe Inter-Language mappings in Section 3.2. However, in this case, the different views of the RAM model are part of the same representation and thus are more closely related than the previous case. For instance, each message view (Sequence Model) describes the functioning of a single operation found within the structural view (design model). Due to this close relationship, navigation between these different representations should be handled in an alternative manner to the standard Inter-Model navigation outlined above.

First, navigation of this type is categorized as Intra-Language (rather than Inter-Language) navigation. This is a distinction whose importance will become clearer in Section 4.1.2. Furthermore, when navigating these connections, the information about the source representation - that is, the view from where the navigation action begins - may

continue to be pertinent after moving to the new representation. For instance, when a user navigates from a structural diagram to the message view which describes a given operation, it is likely beneficial to allow the navigation information for the former model to remain in place in the navigation bar. As such, the element for the message view should be appended to the navigation bar after the element for the structural diagram - without removing any elements from the bar. Conversely, in the case that a user wishes to navigate from one sequence model to another, it is likely not be beneficial to the user to continue to see the navigation information of the previous view. As such, in a similar manner to navigation of Inter-Language mappings, the navigation bar element of the first message view should be removed prior to appending the element for the new message view to the end of the navigation bar.

3.2.2 Requirements

We now formally list the requirements for Inter-Model navigation within TouchCORE:

3.2.1: The navigation system shall show the user how the current model is inter-linked with other models by including related models and model elements in the navigation bar section of a given model.

3.2.2: The navigation system shall allow the user to navigate between models based on the links between models and the links between model elements.

3.2.3: The navigation system shall update the navigation bar automatically to reflect user navigation between models.

3.2.4: The navigation system shall update the navigation bar automatically to reflect changes in the links between models and model elements.

3.2.5: In multi-view models, the navigation system shall show the user how different views of a given model are related to each other.

3.2.6: The navigation system shall allow a user to navigate between the different views of a model.

3.3 Software Product Line Navigation

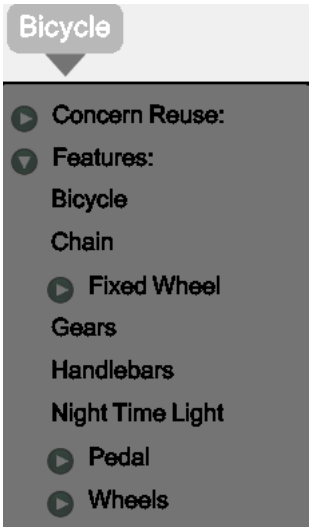
As mentioned in Section 2.3, in addition to the description of a variety of modeling languages, TouchCORE also provides functionality for describing Software Product Lines (SPL) within the tool. As such, in order to ensure that all models described within TouchCORE are able to be navigated effectively, the navigation bar must include functionality for these types of models as well. However, as models in this category represent a different set of functionality from a general model, the navigation bar must also handle them differently. Software product line models form the basis of any modeling in TouchCORE and, as a result, the navigation bar's functionality is not variable for these models as it was in the previous two sections. Instead, the role of the navigation bar is static and treated as a special type of navigation

within TouchCORE. In this section, we describe the differences pertinent to navigation as well as the aims of the navigation bar for these types of models.

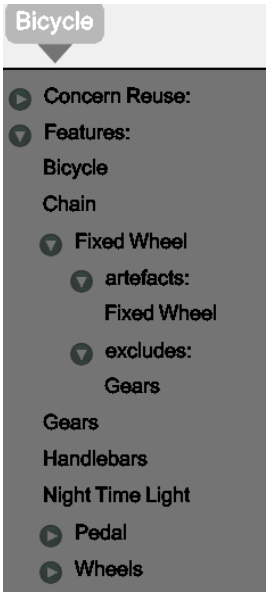
3.3.1 Feature Models

Describing software product lines consists of two types of models, the feature model and the impact model. We begin by outlining the goals of navigation of feature models. Each feature model is populated with a set of features which are connected in a tree structure. An example of this functionality can be seen in Figure 3.5a which shows the navigation bar for the Feature Model seen in Chapter 2. These individual features can themselves be realized by one or many models within the Concern that the feature model is describing. Features may also require other features - meaning that if one feature is selected, the other must be as well - or exclude other features - the opposite, two features cannot be selected at the same time. Thus, navigation within the feature model only requires navigation between individual features. Similar to the outlined functionality for Intra-Model navigation seen in Section 3.1, each of the individual features is listed within the navigation bar. In addition, any instances of exclusion or inclusion between features are listed in a similar manner to that of class hierarchy found within that same section. Thus, within a feature model, the functionality is generally outlined as navigation of an automatically defined language within TouchCORE.

Outside of the elements found within the diagram, the navigation bar for a feature model



(a) The navigation bar for the feature model.



(b) The navigation bar for the feature model with mappings to other features and to Artifacts. The excludes represents a mapping within the model and the Artifact represents a realizing model.

Figure 3.5: Two views of the navigation bar for the Feature Model shown in Figure 2.2.

also includes elements of other models which should be navigable from the feature model. The clearest instance of this would be for navigation to the impact model of the Concern from its feature model. Within the feature model, each of the elements which are present within the impact model for the Concern are listed within their own drop down list in the navigation bar. This list includes all of the impact nodes as well as all of the features which are present within the impact model. If a user clicks on one of the impact nodes, the system should navigate to the view for which that node is the root.

Elements within the feature model can also be realized by one or many models found within the Concern. As a result, it is necessary for a user to easily be able to navigate into models which perform these realizations. In order to provide this functionality, each of the models which realizes a specific feature is listed within a drop down menu for that feature. Clicking on the name of one of the models navigates the user to that model in a similar manner to how navigation to the impact model was described. This functionality is shown for the Feature Model from Chapter 2 in Figure 3.5b.

While navigation away from a feature model to another model has many of the same characteristics as navigation of Inter-Language mappings outlined above, this type of navigation does not take exactly the same form. When navigating Inter-Language mappings, the intention is to replace the current model onscreen with a new model that represents the same part of the system in a different way. However, when navigating from the feature model - which is the launching point for a Concern - the navigation should simulate moving into one part of the Concern - whether that be the goal model of the Concern or a model realizing a specific feature. To represent this visually, rather than replacing the section of the navigation bar of the previous model with a new section describing the new onscreen model, a section should be added to the navigation bar alongside the section for the feature model. In the case of navigating into a realization model for a specific feature, the bar should also add a section reminding the user which feature the model is realizing. The purpose for this functionality will be further outlined in

Section 3.3.3. An example of this can be seen in Figure 3.6 where the Concern section is seen on the far left and the feature section is seen in the middle. The final section represents the realization model for the feature Fixed Wheel.

Figure 3.6: The navigation bar showing all three sections for the realization model Fixed Wheel.

3.3.2 Impact Model

The second model inherent to Software Product Lines in TouchCORE is the impact model. In TouchCORE, each Concern contains a single impact model just as it contains a single feature model. However, unlike a feature model, a single impact model may contain several layouts representing different ways of visualizing the goals of a system. Each goal has a layout in which it is the root goal and this view can contain other goals which interact with the root goal as well as features - derived from the feature model - which can interact with goals within the layout. In order to represent this unusual model, a specific version of navigation needs to be implemented. First, all of the impact nodes within the impact model should be listed within the navigation bar section for this model. This includes nodes which are not present in the layout that is currently on screen, but which are present within the overall model. In addition, all feature nodes which are present in one of the layouts should

also be shown alongside the impact nodes. Together, the impact nodes and all of the features shown within the impact model make up the set of impact elements and are shown within the navigation bar for each layout on screen.

When attempting to navigate within an impact model, there are two situations which may arise. First, the user may wish to see another layout within the model - one for which a different goal than the current root goal is treated as the root for the layout. As such, when a user clicks on a specific goal within the navigation bar, the system should change the onscreen model and show the one for which the selected impact node is the root. Though this changes what is available within the current view, it does not change the components of the navigation bar as they are consistent for all layouts of the impact model. The other type of action a user may wish to conduct is to highlight a feature which is present within the impact model. When a user selects a feature from the navigation bar, that feature is highlighted if it is present within the current layout. If it is not present, nothing occurs as a feature may be included in several layouts so attempting to navigate to it would be unrealistic.

3.3.3 Conflict Resolution of Features

As noted in Section 3.3.1, the feature model represents the base model used to realize a Concern within TouchCORE. As such, navigation into other models within the Concern is facilitated by the individual features in the feature model. Models represent specific features

and thus, when they are displayed, the navigation bar should include information relating to the feature they realize. In the case where the model represents only a single feature within the feature model, this new section is simple - it shows the feature which is being represented without providing any additional options for navigation related to the feature. An example of how this should be realized within the navigation bar can be found in Figure 3.6.

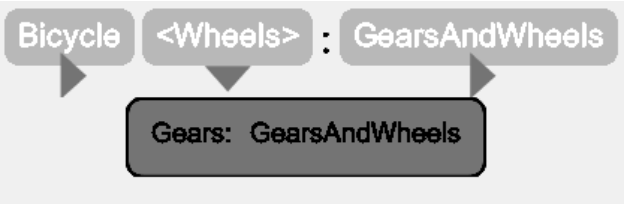


Figure 3.7: The navigation bar showing all three sections for the realization model Wheels. The realization model is associated with two features which are shown in the conflict resolution section of the navigation bar.

However, in certain cases within TouchCORE, a single model may be used to represent multiple features within the same Concern. This is referred to as a conflict resolution model - a term meaning that the model is only relevant if all of the related features are selected and, in that case, overrides any other models related to those features. In the case of a conflict resolution model, the section added to the navigation bar representing the feature is more verbose in its functionality. In this situation, once a user has navigated to the model in question, they may wish to see the other features which that model realizes. To fulfill this need, the new section should contain a drop down menu similar to other sections within the navigation bar. In it is a list of all of the other features which are being realized by

the current model. This can be seen in Figure 3.7 where the Wheels realization model is associated with both Gears and Wheels and thus a conflict resolution section is created. Clicking on one of the features within this list changes the visualization on screen to show the realization of the model from the point of view of the feature which was clicked on.

3.3.4 Requirements

We now formally list the requirements for Software Product Line navigation within TouchCORE:

3.3.1: The navigation system shall generate navigation bar sections for software product line models in the same way that it generates them for models of other languages.

3.3.2: The navigation system shall display realization models that exist within a concern as well as how they are related to the features within the concern's feature model.

3.3.3: The navigation system shall outline the relationship between features and goals found within a concern.

3.3.4: The navigation system shall allow the user to navigate between the feature and goal model of a concern.

3.3.5: The navigation system shall allow the user to navigate to the realization models of a given concern.

3.3.6: When navigating to a realization model, the navigation system shall show the feature to which that model is related.

3.3.7: The navigation system shall facilitate context switches between different views of a realization model based on different features which are related to the model.

3.4 Navigation of Model Reuse

As noted in Section 2.3, TouchCORE is built to facilitate the reuse of models as well as Software Product Lines. For this reason, a fully functioning navigation system for TouchCORE needs to allow users to easily move between models and model reuses. This navigation of reuse can occur in one of two ways - both of which we will outline in the following section.

The first type of reuse is characterized by navigation between models within the same Concern. Often, models used to realize different features within the same Concern contain similar attributes which can be reused to facilitate more efficient description of models. This type of reuse is referred to as a *model extension* and functions very similarly to Inter-Model navigation as described above. When the current model has a model extension available, it

should be shown within the navigation bar of the current model as a subsection specifically for model extensions. When clicked, the navigation system should display the model being extended and the navigation bar should update to show the contents of the new model onscreen as well as the feature from which it is derived (in the case of a conflict resolution, as mentioned in Section 3.3.3, the feature from which it was originally derived should be shown).

The second form of reuse which is pertinent to navigation within TouchCORE is defined by the reusing of models which are not found within the same Concern - which are known as *artifact reuses*. Within an artifact reuse, a specific model from another Concern is used to help define a new model within the current Concern. As such, navigation needs to be defined slightly differently than any Inter-Model navigation seen so far. Much like *model extensions*, when a model has a defined *artifact reuse*, it is shown within its own submenu within the navigation bar section for the current model. However, when that model is clicked and the system navigates to that reused model, the navigation bar should be completely refreshed to show the navigation bar for that new model. The Concern, feature, and model sections within the navigation bar should be populated with new information pertaining to the reused model's information. In addition, an icon - an upper-case 'R' - should be added to the navigation bar to remind the user that they are currently working within a reused model and to remind them to be careful of the changes they make and how they might affect the reusing model(s). An example of this navigation bar setup can be seen in Figure 3.8. When

a user returns to the reused model, the navigation bar should replace the original sections and the reuse icon should be removed - leaving the bar in the same state as it was before the reuse was navigated.

Figure 3.8: An example of the navigation bar functionality in the case of reuse.

When working within reuse situations, there may be a hierarchy of reuses similar to a class type hierarchy in java which needs to be navigated accordingly. In addition, the two types of reuses mentioned above may both be used within the same reuse hierarchy - a *artifact reuse* may in turn be the source of a *model extension* or vice versa. In order to clearly describe how these situations can be navigated, a new type of section should be added to the navigation bar. To fulfill this need, a drop down menu similar to the ones found for other sections of the navigation bar is added below the reuse icon described above. This drop down menu is populated with a last-in-first-out list of all of the reuses which have been navigated to get to the current reuse model. An example of this can be seen in Figure 3.9 where TestDoubleReuse is shown to be reusing Fixed Wheel which is in turn reusing the current model. These previous models can be returned to by clicking on their name within the section. When a user returns to a previous model, the system should remove models from the reuse section accordingly and this system should generalize to any number of links within the reuse hierarchy.

Figure 3.9: An example of the navigation bar functionality in the case of double reuse. One can click on one of the previous models from within the reuse section.

3.4.1 Requirements

We now formally list the requirements for software reuse navigation within TouchCORE:

3.4.1: The navigation system shall outline any instances of model extension or model reuse associated with a model in that model's navigation bar section.

3.4.2: The navigation system shall update to automatically reflect any newly associated instances of extension or reuse.

3.4.3: The navigation system shall allow the user to navigate into an instance of extension or reuse.

3.4.4: The navigation system shall allow the user to return to a model from an extension model or a reused model.

3.5 Summary

In this chapter, we outline the intended functionality of a generic navigation system. We introduce the idea of several types of navigation, namely: Intra-Model navigation, Inter-Model and Inter-Language navigation, Software Product Line navigation, and the navigation of reuse relationships. In addition, we outlined a set of formal requirements for each type of navigation. In the following chapter, we outline the implementation details of the generic system in TouchCORE which is used to implement these navigation concepts.

Chapter 4

Implementation

Having outlined the functionality intended for the generic navigation bar in the previous chapter, we now turn to the implementation details of the navigation system within TouchCORE. In this chapter, we outline the metamodel of the navigation system as well as the path it took from its original description before delving into the specifics of how each of the aforementioned requirements is satisfied by the system.

4.1 Navigation Metamodel

4.1.1 Navigation Metamodel Evolution

In this section, we introduce the original outline of the navigation metamodel created by Ali *et al.* [2]. This metamodel, along with the preliminary implementation details outlined by Ali

et al. provide the foundation from which our implementation has grown. We also overview the changes made to that original conception of the metamodel in order to successfully create a navigation system in TouchCORE which conformed to the specifications outlined in Chapter 3. The details of the purpose of each of the metamodel elements is saved for Section 4.1.2.

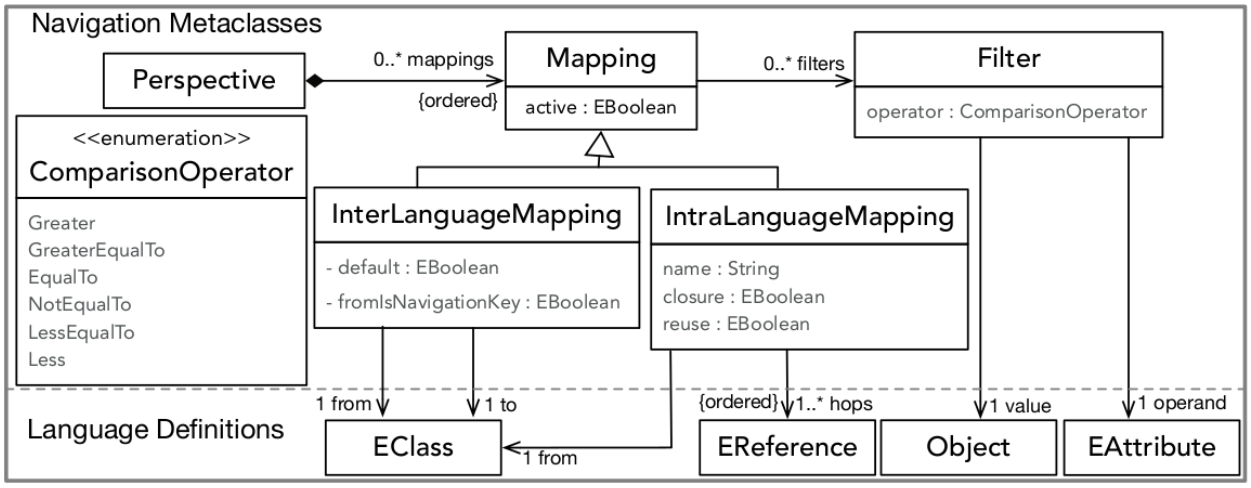


Figure 4.1: The original description of the navigation metamodel as outlined by Ali *et al.* [2].

Figure 4.1 shows the original conception of the metamodel. As with the later implementation of the metamodel, the original metamodel was described within the context of the Eclipse Modeling Framework (EMF) and in accordance with that framework, is expressed within the metamodeling language ECore. Therefore, each of the concepts shown within the model can be assumed to conform to ECore conventions - such as each of the classes shown being instances of *EClass* etc.

In the following section, we will outline the final version of the metamodel - which can be seen in Figure 4.2 - in detail. Prior to this, we outline the changes made to reach this final state both in order to describe our design decisions and to demonstrate the scope of the evolution.

Some of the structure of the metamodel has remained through the editing process. First, the Perspective is still the central component and is still composed of a set of mappings - though the name has been changed to **NavigationMapping** in order to more clearly differentiate between language mappings. The navigation mappings continue to be split into two groups - Inter-Language and Intra-Language Mappings - with the **IntraLanguageMapping** end points remaining consistent between the two metamodels. Finally, several of the attributes present within the original metamodel - such as **name**, **closure** and **reuse** within **IntraLanguageMapping** - persist into the final version.

However, beyond some of the central classes mentioned above, much has changed within the metamodel to more accurately describe the functionality of the navigation bar within TouchCORE. First, Inter-Language mappings are now associated with CORELanguageElementMapping - a concept originally defined within the TouchCORE perspective metamodel - in order to more closely tie their functionality to the description of the Perspective itself. In addition, the class **InterLanguageMappingEnd** has been included to replace the functionality of the **from** and **to** *EClass's* within **InterLanguageMapping**. This new *EClass* connects to the MappingEnd which was

again initially defined within the Perspective metamodel within TouchCORE. Overall, these first two changes have the affect of directly tying the definition of Inter-Language mappings to the defined language mappings within a Perspective.

Furthermore, Intra-Language mappings now also have several more attributes which are available to allow a language designer to exert more control over the functionality of the navigation bar for their given language or perspective. Finally, the Filter system which was outlined in the initial metamodel has been removed as we decided that other features were more important to the functionality of the navigation system. This system includes the Filter class and ComparisonOperator enumeration along with the associations made between Filter and several other classes. The Filter system is categorized as possible future work.

4.1.2 Navigation Metamodel Implementation

Having described the evolution of the navigation metamodel in the previous section, we now outline, in detail, the final implementation of the metamodel. We describe the function of each of the *EClasses* and their associations as well as the relationship between the navigation metamodel and the Perspective metamodel within TouchCORE. As noted above, this final implementation has also been defined in ECore and, as such, all conventions noted above still apply. The final implementation of the navigation metamodel can be seen in Figure 4.2.

The central component of the navigation system is the Perspective - represented in the metamodel by the **COREPerspective** *EClass*. As can be seen within the metamodel,

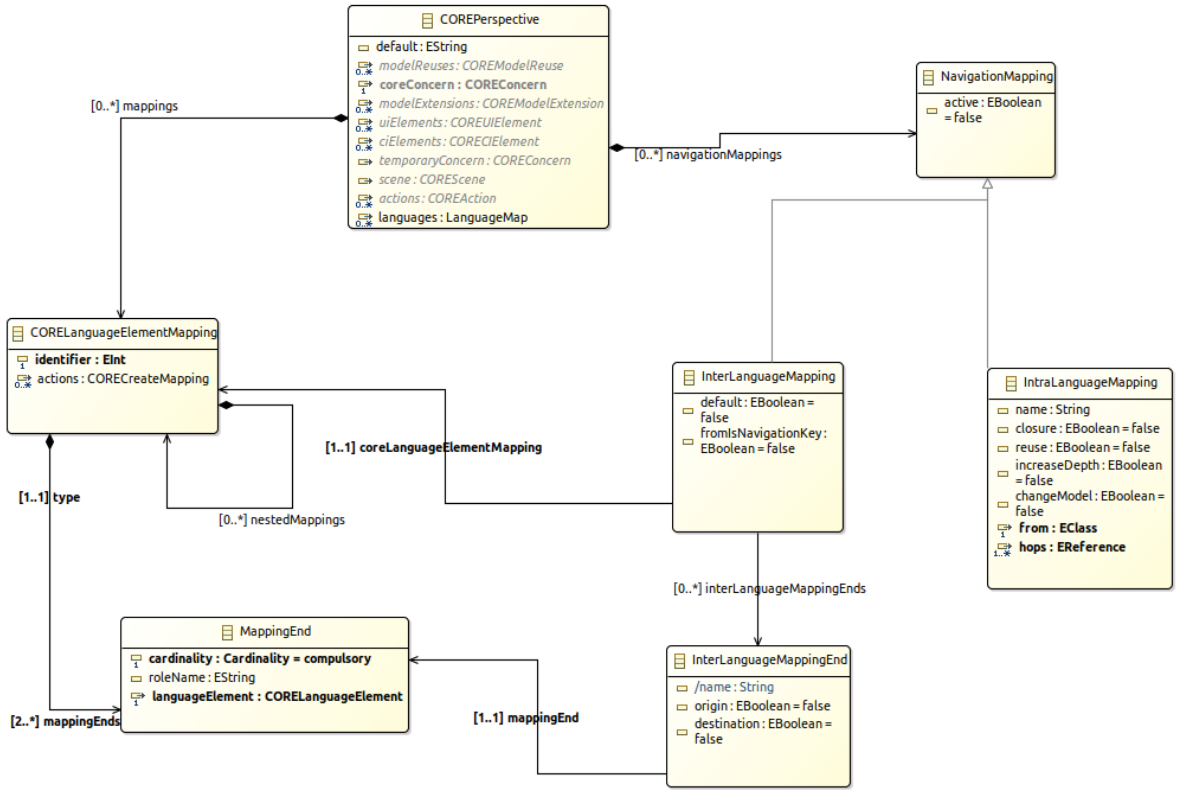


Figure 4.2: The metamodel for the navigation system within TouchCORE.

COREPerspective contains several attributes and associations which are held over from its inclusion in the Perspective portion of the overall TouchCORE metamodel. Consequently, these can be disregarded in this context.

Composing the Perspective is the set of navigation mappings which detail the navigable links for that Perspective. These Navigation Mappings are split into two categories - Intra-Language mappings and Inter-Language mappings - mirroring the two different types of navigation which were outlined in Chapter 3. Each of these can be toggled on or off with

the **active** attribute within **NavigationMapping**. When **active** is false, the mapping does not appear within the navigation bar but does not require the user to completely remove it from the system to achieve this functionality.

Intra-Language mappings model links between elements within the same language. More specifically, they are used in cases in which the navigation is carried out through associations found within the metamodel for the language of the given model. The **from** *EClass* functions as the origin of a mapping and each of the *EReferences* within **hops** are used to traverse through the metamodel to the destination of the navigation - the *EClass* which is reached by traversing the final *EReference* within hops. The hops are, naturally, chained together so that the destination of one hop is treated as the origin of the next.

The **IntraLanguageMapping** *EClass* also includes several attributes that allow language designers more control when defining the mappings for a given language. First, the **name** attribute can be used to define how the mapping is categorized within the navigation bar. The **closure** boolean is used in the case that there is more than one level of mapping that needs to be traversed. For instance, when navigating the class hierarchy within a class diagram, there may be several levels of ancestors for a given class. If the **closure** field is set to true, the system traverses the whole class hierarchy and shows all of the super-classes from all levels of the inheritance structure as destinations for the mapping. If the **closure** field is false, the destination of the mapping is the direct parent of the origin class. The **reuse** field is used in order to alert the system that the mapping

relates to an instance of reuse and thus a context switch must occur when that mapping is traversed. The **increaseDepth** field is used in the case of a multi-view language to signal that traversing that mapping should add a new section onto the end of the navigation bar for the new view without removing the current section from the bar. Finally, the **changeModel** attribute is also used in the case of multi-view languages to alert the navigation bar to stop displaying navigation mappings after the current one as they are related to other views within the model.

CORELanguageElementMapping is an *EClass* which is present within the TouchCORE Perspective metamodel to describe Perspective mappings between elements of different languages. Due to the nature of Inter-Language navigation defined in Section 3.2, we have chosen to derive Inter-Language mappings from the already defined Perspective mappings in order to ensure consistency and ease of definition for language designers. As such, a one-to-one association is included in the navigation metamodel between **InterLanguageMapping** and **CORELanguageElementMapping** to facilitate this connection. In addition, at run-time, when the system is creating instances within the navigation bar, the **identifier** attribute is used to facilitate the conversion from connections between *EClasses* to connections between *EObjects*. This will be explored in more depth in Section 4.2.2

Inter-Language mappings, in contrast to their Intra-Language counterparts, are used in instances where traversal does not occur by moving through the metamodel of a given

language. Instead, the source and destination of an Inter-Language mapping are derived from different metamodels or from the same metamodel being used in different roles within a Perspective. In addition, unlike in the case of Intra-Language mappings where traversal was unidirectional - that is, the source and destination of the mapping could not be inverted - Inter-Language mappings can be bidirectional. These two changes require a different method for defining the ends of a Inter-Language mapping.

This task is now completed using the **InterLanguageMappingEnd** *EClass* which is associated with **InterLanguageMapping**. Due to the fact that Inter-Language mappings may have more than two ends - that is, there may be one source which is connected to several destinations - the multiplicity of this association is zero-to-many. Each of the Inter-Language mapping ends is connected to a single **MappingEnd** which is another *EClass* previously defined within the Perspective metamodel. Each **MappingEnd** contains an association to a **LanguageElement** which serves to define the component found at one end of the mapping. These language elements are used to determine whether elements found within the Perspective should be included within a given mapping. It also contains the **roleName** field which defines the role that the language - and by extension the component at one end of the mapping - is performing in that context. The final attribute found within this class, **cardinality** is not pertinent in this context.

The **InterLanguageMappingEnd** also includes several other attributes which are used to describe individual mappings in more detail. First, the **name** attribute is derived from

the **roleName** and **languageElement** within the associated **MappingEnd** and is used to determine the name of the mapping when shown within the navigation bar. The **origin** and **destination** attributes are both concerned with the bidirectionally of a specific mapping. In some cases, language engineers may wish for a specific mapping to function only as the source or destination of a mapping. If the **destination** field is set to false for one end of a mapping, that end is not treated as the destination for that mapping. Likewise, if the **source** field is set to false, that mapping end is not treated as the origin of a connection within the navigation bar. This allows the language engineer to define what elements should be navigable in what direction.

4.2 Implementation Details

Having outlined the metamodel which is used as the basis for the navigation system within TouchCORE, we now turn to the implementation details of the system. Much of the implementation is focused on meeting the specifications outlined in Chapter 3 and is thus not again described in detail. Instead, we outline how users may interact with the aforementioned metamodel to obtain this functionality.[1]

[1]The full source code for TouchCORE and, by extension, the generic navigation bar can be found at: https://bitbucket.org/mcgillram/workspace/projects/TC.

4.2.1 Definition of Navigation Concepts in TouchCORE

In Section 3.1, we introduced the need for a system to specify which elements within a language are displayed within the navigation bar and thus navigable from and to when viewing a model. To facilitate this function, we turn to the metamodels of individual languages defined within TouchCORE and look to associate them with concepts found within the metamodel for the navigation system. The specification begins with the metaclass for an individual model for that language - with one or multiple navigation mappings being specified using the model metaclass as an origin. From this point, the definition of mappings forms a tree structure, with the destination of an individual navigation mapping forming the possible origin of other mappings. An example of this functionality can be found within the Class Diagram language. The initial modeling concept is the metaclass for the *Class Diagram* itself which is linked via an Intra-Language mapping to the metaclass for *Class* within that language. The *Class* metaclass may, in turn, function as the origin of several other mappings - possibly to the metaclass for *Operation* or to itself to facilitate the navigation definition of a class hierarchy.

This tree structure can be implemented for Inter-Language mappings as well. For instance, the Class Diagram metaclass may be treated as the origin for an Inter-Language mapping whose destination is the metaclass for a Use Case Diagram within the Use Case language or a similar mapping can be made from an individual Class to the Actor metaclass within a Use Case Diagram. However, there are a few differences between the

functionality of Intra and Inter-Language mappings. First, as noted in Section 4.1.2, Inter-Language mappings are defined in relation to Language mappings within a Perspective and, as such, are not defined directly as navigation mappings but are derived by the TouchCORE system from those Language mappings.

The second key difference arises from the tree structure of navigation mappings. As noted above, metaclasses for a given language are arranged in a tree structure starting with the metaclass for the model itself. Due to this, any metaclass which a language designer wishes to include within the navigation system needs to be added to the tree via navigation mapping with a path back to the root metaclass. However, whereas Intra-Language mappings can be used to specify any edge within this tree, Inter-Language mappings are used only to define edges ending in leaf nodes. This is due to the fact that Inter-Language mappings necessarily have their origin and destination be found within different models. As such, any mapping beginning from a metaclass on the destination end of an Inter-Language mapping is a part of the navigation tree for that model and thus should not appear within the current tree.

The definition of the navigation mappings is currently performed via a manual encoding process by a language designer. The language engineer has full control over the navigation and specifies the Intra-Language mappings for each language found within a Perspective as well as the Inter-Language mappings connecting them in cases in which the Perspective contains multiple languages. These are then automatically encoded into the CORE description of the given language(s) and Perspective. When a model of this language is

encountered within TouchCORE, the navigation mappings, along with the metamodel for each language, are used to populate the navigation bar - a process which we outline later in this section. In the future, we wish to implement this system with a graphical user interface in order to allow for easier definitions of mappings but as yet encoding cannot be done in this manner.

As noted in Section 3.3, in order for navigation in TouchCORE to be fully specified, navigation for concerns needs to be available regardless of the specific languages used in a given instance. As such, rather than allowing for user definition of navigation mappings for feature and goal models, we hard-code these navigation mappings into TouchCORE so there is no variation between how they appear in the navigation bar between different concerns. While the specifics used for specifying feature and goal models are not outlined here as we believe it would not provide any greater understanding of the use of TouchCORE, we state that the hard-coded mappings are sufficient to conform to the specifications in Section 3.3 and we test them accordingly as stated in Chapter 5.

4.2.2 Run-Time Generic Navigation

Once a language engineer has defined the pertinent navigation concepts for a given language or Perspective, the final question regarding how generic navigation in TouchCORE is completed centers on how these concepts translate into actionable concepts in the navigation bar. In this section, we detail how this step of the process is performed.

When a new model is navigated to on-screen - whether through the navigation bar or by other means - the system calls the navigation system with the specific model to be displayed. The system then checks the language of the model as well as the current Perspective (if applicable) and obtains the related navigation mappings which were previously defined by a language engineer. In a similar fashion to the EMF standards used in defining the metamodel for TouchCORE and for the individual languages within TouchCORE, each of the models shown, along with their model elements, is defined with EMF specifications. Model elements are defined by *EObjects* which function as instances of the individual *EClasses* defining the metamodels. Therefore, when a new model is called for, the system queries the *EObjects* that make up the model and compares their *EClasses* to those found within the navigation mappings. In the cases that connections between *EObjects* correspond to connections found within navigation mappings, the destination of that connection is added to the navigation bar. In this way, the tree structure created for an individual model mimics the tree structure created for the navigation mappings.

The tree created for a model is then populated to the navigation bar to conform to the specifications outlined in Chapter 3. Each node translates to a single navigation element within the navigation bar and mappings result in submenus which can be opened and closed depending on the user's needs. Each different navigation mapping has its own submenu below a pertinent node when there is at least one instance of that mapping present beginning with the *EObject* related to that node. The tree is expanded or contracted to reflect changes made

in the model - nodes are added to the tree when pertinent elements are added to the model and vice versa - so that the navigation bar continues to reflect the important concepts of the current on-screen model.

When a user navigates away from a model, the sections present within the navigation bar are updated accordingly - they are removed if no longer applicable or allowed to remain when they still hold bearing on the current model. An example of when the previous section would not be removed is the case of moving from a feature model to a realization model. However, in cases where more than one navigation bar section is present, only one can be displayed at a time so as not to clutter the screen and provide for streamlined use. Clicking on the name of another model - or of an element from another model - within the navigation bar automatically changes the model shown and updates the navigation system accordingly. Finally, a fully functioning back button is provided so that users can return easily to models previously shown.

4.2.3 Architecture Overview

Having introduced the run-time functionality of the new navigation in TouchCORE, we turn to an overview of the architecture of the navigation system. An architectural diagram of the system showing its important parts can be seen in Figure 4.3. Within the diagram, we do not focus on any TouchCORE concepts that exist outside of the navigation system though there is naturally interaction between concepts found within the diagram and those not. A more

detailed pseudo-code description of several of the main algorithms related to the navigation bar can be found in Appendix B.

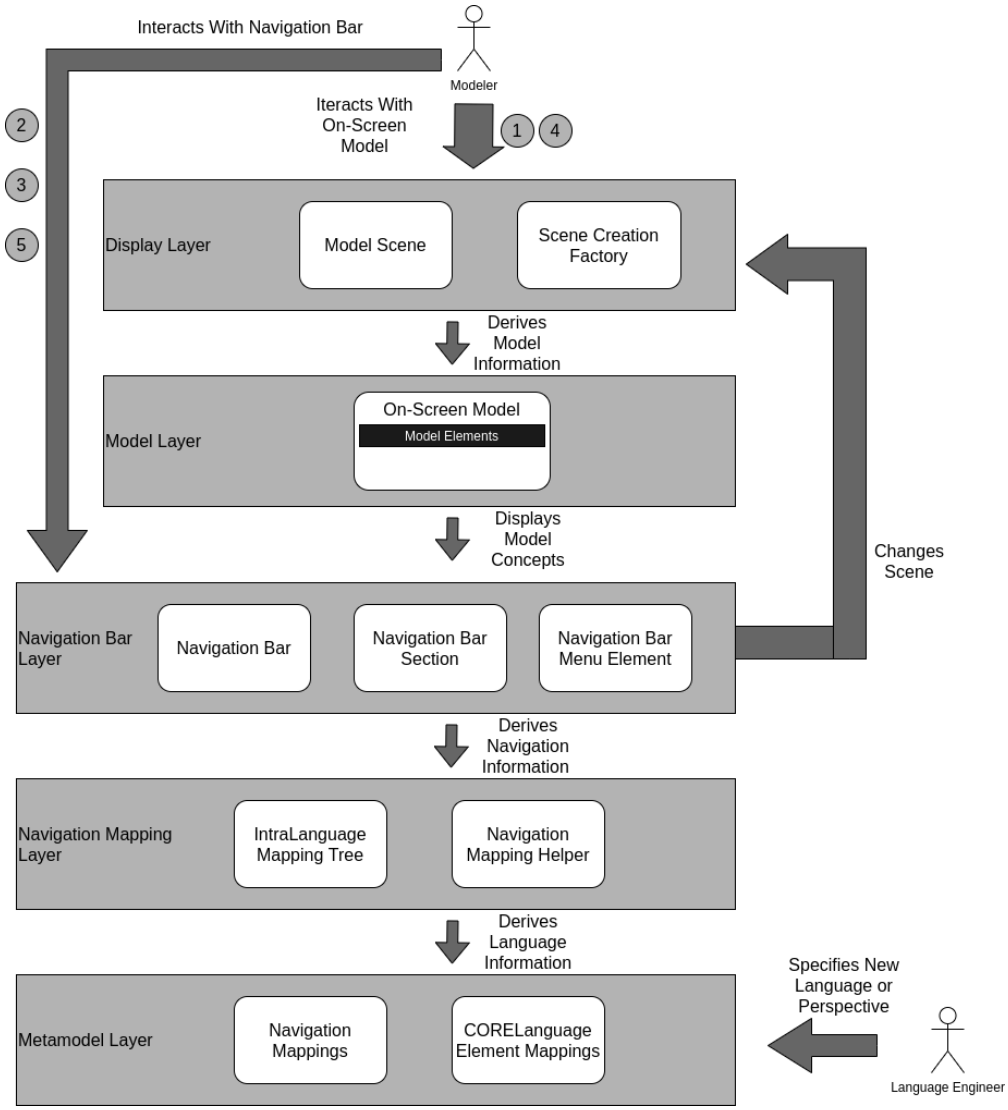


Figure 4.3: An architecture diagram displaying the main components of the navigation system. The numbered circles represent the initiation of algorithms found within the pseudo-code specified in Appendix B.

When a modeler wishes to use TouchCORE, they interact with the current model in its

Model Scene which is a class made to display a model of that type in TouchCORE. If they call for a scene to be changed, the *Scene Creation Factory* creates a new scene (if one has not yet been created for that model) or calls for an existing scene to be displayed. These two classes make up the display layer of the navigation system as they directly relate to the model concepts which are shown to the modeler. They derive their information from the Model Layer which contains the current model which is open as well as its model elements.

The Navigation Bar Layer lists the information of the current model as well as its mappings to other models as outlined in Chapter 3. This layer contains the *Navigation Bar* component which houses the main navigation bar functionality as well as the back button system. The navigation bar is composed of *Navigation Bar Sections* which represent the individual components for a model, feature, or concern. These are added and removed when the on screen model is changed by the modeler. The individual sections are made up of *Navigation Bar Menu Elements* which house the individual lists of elements displayed for the component related to the section. For instance, for a given Class Diagram, the Menu Element would house the information on classes, operations, and any Inter-Language links to other models. A modeler may also interact with the navigation bar layer by opening the navigation bar section, selecting one of the elements in one of the lists, or clicking on the back button. If one of these inputs causes the scene to be changed (such as selecting an Inter-Language link), the Navigation Layer calls the display layer to perform these changes in the same way as if it were from a user input directly in the scene.

The Navigation Layer derives the information on the contents of the sections and menu elements from the Navigation Mapping Layer. When a new model is shown on screen, the *Navigation Bar* calls the *Navigation Mapping Helper* to create a new *IntraLanguage Mapping Tree* which uses the methodology outlined in Section 4.2.2 to create a Navigation Mapping tree. This tree will then be returned to the Navigation Bar Layer and added to the pertinent menu element. The Navigation Mapping Layer derives the information used to create the tree from the Metamodel Layer which houses the information on both the *Navigation Mappings* defined for the current language or Perspective as well as the *CORELanguage Element Mappings* which are used to facilitate the addition of Inter-Language Mappings to the tree. When a language designer defines a new language or Perspective for use within the navigation bar, they are interacting with the Metamodel Layer by creating new Navigation Mappings and CORELanguage Element Mappings which can then be used as part of the navigation system.

4.2.4 Navigation Algorithm Description

We now describe one of the described pseudo-code algorithms - found in Appendix B - in greater detail to give a clearer outline of the functioning of the system. Algorithm 1 outlines the system's reaction to a user selecting a new model to be shown on the screen. First, the modeler calls for the new scene through one of the available buttons which is reacted to by the listener associated with that button (line 1). From this point the *Scene Creation Factory*

within the Display layer is called upon to either create a new *Model Scene* or obtain one which has already been created for the specified model (line 2). The factory then displays this scene which will contain various model elements related to the associated model (line 3).

From this point, the *NavigationBar* itself is called and a new *Navigation Bar Section* is added to the list of sections housed there (line 4). Next, the section is added to the screen and initialized with a *Navigation Bar Menu* which will be used to house the actual model information for that section (line 5). This information will be associated with a *Navigation Bar Namer* which will be called in the case that a user interacts with one of the elements within the section (line 6). Finally, a listener is created for the section to alert the system of any actions made on the navigation bar section (line 7).

4.3 Summary

In this chapter, we outline both the evolution of the navigation metamodel as well as its current implementation within TouchCORE. Later, we describe the method of defining the navigation concepts for a new language or Perspective as well as the run-time details of the navigation bar within TouchCORE. Finally, we introduce the architecture of the navigation system as well as the details of one of the algorithms used to implement the functionality of the navigation system. In the next chapter, we describe the testing system used to validate the navigation system in TouchCORE.

Chapter 5

Testing

With our implementation of a generic navigation system fully outlined, we now turn to the problem of testing the tool in order to ensure it conforms correctly to the needs of users within TouchCORE. Due to the fact that the navigation system of a modeling tool such as TouchCORE relies largely on user interaction with a UI, a conventionally defined testing methodology based around test code such as JUnit[1] does not provide the necessary confidence that we have reached a proper implementation. Instead, we turn to a set of human tests which are executed within the TouchCORE client.

[1] https://junit.org/junit5/

5.1 Test Suite

We aim to create a suite of tests able to assess the quality of all of the features and situations found in Chapter 3 as well as validating many of the edge cases which may arise from situations outside the clear bounds defined in that chapter. However, we naturally are not able to include all possible situations that may arise within TouchCORE as the set of all modeling tasks would be infinite in size. Instead, we list all of the tests performed to test validity in Appendix A to provide readers with a full knowledge of the situations tested and not tested.

Each of the tests performed is listed in the same format. An initial situation is provided which describes the system in its initial state prior to the change we wish to test. Then, we introduce a specified change to the system. Both of these are described in only the pertinent details to allow a user to introduce some randomness into the testing phase and possibly catch unanticipated errors. The final column for each test represents the expected change to the system as a result of the user input. This output is found true or false depending on whether the situation presented in the first two columns leads to the expected situation in the final column.

To clarify the process of testing the tool, we look to an example test and go step-by-step from justifying the test to the expected output. Test 33 is attempting to assess the ability of the navigation system to handle the creation of new realization models (artifacts) which conform to a multi-language Perspective. This represents a core functionality within

TouchCORE and is used to partially confirm the success of the tool at meeting several of the requirements outlined within Chapter 3 - specifically requirements 3.1.1, 3.1.5, 3.2.1, 3.3.5, and 3.3.6. When executing the test, we must begin by viewing the Feature Model of a given concern as that will be the location from which new realization models can be created. We then create a new artifact of a Perspective with multiple languages through any means provided within TouchCORE as the response should be consistent no matter how the test is initiated. We then check the output to see if the following reactions have occurred: the onscreen model changes to one of the newly created models, a feature section has been added to the navbar for the associated feature, a section within the navigation bar has been created for the new realization model, and when the model section is opened there should be Inter-Language mappings automatically shown which connect to each of the other Perspective models which were created. If each of these reactions has occurred, we consider the test successfully completed. We follow a similar path for all other tests listed in Appendix A.

Table 5.1 shows each of the requirements as they were formulated in Chapter 3 along with each of the tests which we used to ensure the final product conformed to each of the goals of the system. As can be seen in the table, each of the formulated requirements was tested by at least two of the tests listed in Appendix A. In addition, several of the requirements - such as 3.1.3 and 3.3.6 - were assessed by a large number of tests due to the fact that many actions available within TouchCORE impact the completion of those requirements. In

Table 5.1: An overview of each of the requirements defined within Chapter 3 along with which tests were used to ensure their completion.

Number	Related Topic	Requirement	Tests Related
3.1.1	Intra-Model	The navigation system shall be able to outline what concepts exist within a given model while also being concise enough to not overwhelm the user with information.	1, 7, 8, 33
3.1.2	Intra-Model	The navigation system shall make clear how elements of a model are related to each other within the structure of the navigation bar section for that model.	3, 14, 15, 16
3.1.3	Intra-Model	The navigation system shall automatically update the navigation bar section of a model to reflect changes made to the model such as adding elements, deleting elements, or changing their contents.	2, 3, 4, 5, 6, 13, 14, 15, 16, 19, 20, 21, 41, 50
3.1.4	Intra-Model	The navigation system shall allow the user to locate a model element within the current model.	17, 18
3.1.5	Intra-Model	The navigation system shall automatically generate navigation bar sections to reflect mappings defined by language designers.	1, 7, 8, 33
3.2.1	Inter-Model	The navigation system shall show the user how the current model is inter-linked with other models by including related models and model elements in the navigation bar section of a given model.	33, 38
3.2.2	Inter-Model	The navigation system shall allow the user to navigate between models based on the links between models and the links between model elements.	34, 36, 37
3.2.3	Inter-Model	The navigation system shall update the navigation bar automatically to reflect user navigation between models.	34, 36, 37
3.2.4	Inter-Model	The navigation system shall update the navigation bar automatically to reflect changes in the links between models and model elements.	35, 38
3.2.5	Multi-View Models	In multi-view models, the navigation system shall show the user how different views of a given model are related to each other.	50, 51, 52, 53
3.2.6	Multi-View Models	The navigation system shall allow a user to navigate between the different views of a model.	51, 52, 53

Table 5.1

3.3.1	Software Product Lines	The navigation system shall generate navigation bar sections for software product line models in the same way that it generates them for models of other languages.	1, 40
3.3.2	Software Product Lines	The navigation system shall display realization models that exist within a concern as well as how they are related to the features within the concern's feature model.	7, 8
3.3.3	Software Product Lines	The navigation system shall outline the relationship between features and goals found within a concern.	9, 10, 40, 41
3.3.4	Software Product Lines	The navigation system shall allow the user to navigate between the feature and goal model of a concern.	39, 43, 44
3.3.5	Software Product Lines	The navigation system shall allow the user to navigate to the realization models of a given concern.	7, 8, 11, 12, 17, 33
3.3.6	Conflict Resolution	When navigating to a realization model, the navigation system shall show the feature to which that model is related.	7, 8, 11, 12, 33, 46, 47, 48, 49
3.3.7	Conflict Resolution	The navigation system shall facilitate context switches between different views of a realization model based on different features which are related to the model.	46, 47, 48, 49
3.4.1	Model Reuse	The navigation system shall outline any instances of model extension or model reuse associated with a model in that model's navigation bar section.	24, 26, 29
3.4.2	Model Reuse	The navigation system shall update to automatically reflect any newly associated instances of extension or reuse.	22, 23, 28
3.4.3	Model Reuse	The navigation system shall allow the user to navigate into an instance of extension or reuse.	24, 26, 29
3.4.4	Model Reuse	The navigation system shall allow the user to return to a model from an extension model or a reused model.	25, 27, 30, 31, 32

addition, most of the test are listed under more than one requirement from Chapter 3. This is due to the fact that actions in TouchCORE often impact several of the goals at once.

We believe that - given the success of all of the tests within the suite - we can be confident that the system is able to conform to all of the requirements which we have outlined. Thus, given that success, our testing expectations have been met. There are likely bugs that remain in the system; however, we feel that, given that these tests cover a vast majority of use cases, the bugs that remain are within the bounds of acceptable completion.

All testing was performed on the final version of the product to ensure that no changes made to the navigation system would result in earlier situations no longer working correctly. Tests were performed by the author of the thesis - and therefore the tool - and were all found to be successful. Therefore, we find that all of the requirements outlined in Chapter 3 are fulfilled within the current version of the navigation system.

In the future, we intend for a more automated version of the testing system to be implemented within TouchCORE so that any changes can be immediately checked to see if they cause any of the situations outlined to no longer perform correctly. Several frameworks exist to facilitate GUI testing within Java such as Rapise[2] and Abbott[3] which may help implement automatic testing for the navigation system. However, we believe that this is currently outside the scope of this thesis and thus leave it as future work.

[2]http://www.inflectra.com/Rapise/
[3]http://abbot.sourceforge.net/doc/download.shtml

5.2 Summary

In this chapter, we outline the testing methodology used to validate the newly implemented TouchCORE generic navigation bar. In addition, we outline how each of the tests performed within the testing suite were used to verify each of the requirements listed within Chapter 3. In the next chapter, we outline a survey of similar modeling tools to compare our navigation bar to other systems performing similar functions.

Chapter 6

Related Work

In this chapter, we outline work related to our topic of generic navigation of software models. As this thesis is largely based upon the implementation of a system within TouchCORE, we feel that a traditional review of the literature would not provide a sufficient understanding on the current state of the art on this topic. Instead, we opt for a survey of several other modeling tools and the ways in which they perform navigation within and between different models.

We searched online for several tools which were built to provide full support for modeling UML diagrams. UML [18] is a widely used standard for modeling software languages and provides definition for 14 different common software models. While UML does not extend to the possibilities of DSMLs as is the intention for the TouchCORE tool, it does provide standardization for many of the languages currently available in TouchCORE - including

many of the languages used to test the system in Chapter 5. As such, we believe that comparing our tool to popular UML modeling software products would provide a good allegory to TouchCORE.

In order to find suitable tools we searched Google for popular modeling tools and chose suitable examples using the following criteria.

- The tool must provide support for most if not all of the models defined by UML. Tools with very limited modeling capacity - such as those which are tailored only to a single model - would not be applicable as we are interested in navigation between models.
- The tool must be available for free or provide a demo version for testing.
- The tool must be able to be installed on modern operating systems and must be somewhat supported at the time of writing this thesis.
- The tool must provide for the creation of multiple models at once - preferably in different languages.

We were able to find several tools that met the description and chose four to use for our survey as they were referenced often when performing our search. Namely, the four tools are: **StarUML** [4], **MagicDraw** [5], **Visual Paradigm** [19], and **Sparx Enterprise Architect** [20]. In addition to these, the most mentioned tool we saw online was **ArgoUML**. However, when attempting to use the tool we found that many of the reputable download links were no longer supported or were broken. In addition, the tool has not been formally

supported since 2015. As such, it did not meet the criteria for our survery. However, due to its prevalence in online literature, we felt we should mention the tool rather than omitting it completely.

6.1 Survey of Navigation Systems

We now outline the results of our survey of navigation systems in software modeling tools.

6.1.1 StarUML

StarUML [4] is an open source tool which provides functionality for modeling UML diagrams, with support available for all 14 UML diagrams. The tool is compatable with UML 2.x.

Navigation in StarUML takes place through the model explorer shown on the right of the screen. Double clicking on any of the models within StarUML opens that model within the tool. There is a hierarchy within the explorer - models are shown to contain model elements which in turn may contain other elements and this is shown through drop down menus in the explorer. However, all elements are shown in this way and cannot be filtered out as is the case for attributes in Class Diagrams within the TouchCORE tool.

Navigation between models is supported. A user may select a different model to be shown on screen in the explorer and it is then displayed. However, this is the only way of navigating to another model. Clicking on one of the elements does not allow for navigation to that model. One must click 'select in diagram' to perform this action. In addition, the

only instance of inter-related models seems to be between operations in a Class Diagram and related sequence diagrams which is shown within the navigation tool. Other links between models, such as was included within Perspectives in TouchCORE, are not available within the navigation system. In addition, there is no back button available for moving quickly to previous models.

6.1.2 MagicDraw

MagicDraw [5] is a commercial tool which can be used to model diagrams within UML 2.x. Support is again available for all UML diagrams.

Navigation in StarUML takes place through the model explorer in the top left of the screen and clicking any model in the explorer opens that model. There is support for intra-model navigation through drop down menus in the explorer, with concepts such as attributes and operations shown underneath their classes in a class diagram. However, the actual elements such as classes, actors and use cases are all shown on the same level as the diagrams they are in and it does not show containment within the model. There does not seem to be any system to filter out unwanted element types such as attributes within a given diagram.

When navigating between models, one can easily change to another model within a project through the explorer. However, clicking on an element belonging to a specific model within the explorer creates a pop-up with its description and qualities rather than navigating to that model or highlighting the element. Models within the tool are also not explicitly linked

in any way and so models and elements cannot be navigated based on inter-related concepts and are only shown in the explorer within their own model description. One exception to this is sequence diagrams, which can be navigated to through from their associated operation within a class diagram. In addition, there is no back button support to return to a previously shown model quickly.

6.1.3 Visual Paradigm

Visual Paradigm [19] is a commercial tool which can be used to model diagrams within UML 2.x. Support is again available for all UML diagrams.

Navigation within a Visual Paradigm project is available in several parts of the UI with the most verbose being through the model structure tab. This causes an overlay on the screen displaying all of the models within a project as well as all of the elements present in those models. The elements shown within the overlay do not have an explicitly defined structure such as showing class hierarchy within a Class Diagram. There is no way of dictating what types of elements should be listed within the model structure tab.

Navigation between models can be performed by clicking on one of the elements within the model structure overlay. However, clicking on one of the elements of the models brings up its properties but does not bring up the model itself. There seems to be no inter-linking between models in Visual Paradigm and thus models cannot be navigation based on their inter-relations as they can be in TouchCORE. As with the first two systems, there is no

explicit back button.

6.1.4 Sparx Enterprise Architect

Much like the previous two tools, Sparx Enterprise Architect [20] is a commercial tool which provides support for all diagrams specified in UML 2.x.

Navigation in Sparx Enterprise Architect takes place through the navigation bar on the right of the screen. Navigation is handled differently depending on the language. For instance, class diagrams have all of the classes listed with individual drop down menus for the attributes and operation. However, in the case of a Use Case Diagram, drop down menus are based on the different types of elements present such as Actors and Use Cases.

Navigating between models can be performed by clicking on a model within the explorer. In addition, in much the same way as in TouchCORE, clicking on an element results in that element being highlighted and, in the case that the element resides in a different model, changes the on-screen model to display the model which contains the given element. As with the Visual Paradigm, there does not seem to be support for inter-connections between models and thus navigation cannot be performed via this type of relation. Finally, as with all of the previous tools, there seems to be no back button functionality.

Table 6.1: An overview of the survey of navigation tools in related software modeling products.

System	Navigation Tool	Intra-Language System	Navigation available via models	Navigation available via model elements	Navigation available between inter-related models	Filtering of element types	Back Button
StarUML	Model explorer on right of screen	Yes	Yes	No	Yes – in specific cases	No	No
MagicDraw	Model explorer in the top left of the screen	Yes	Yes	No	Yes – in specific cases	No	No
Visual Paradigm	Model structure tab	Yes	Yes	No	No	No	No
Sparx Enterprise Architect	Navigation bar on the right of the screen	Yes	Yes	Yes	No	No	No
TouchCORE	Navigation bar at top of screen	Yes	Yes	Yes	Yes	Yes	Yes

6.1.5 Overview

We display an overview of the results of our survey in Table 6.1. Within the four tools surveyed, we find many commonalities regarding the navigation system provided to users. Much like the TouchCORE navigation system, all of the tools have some type of hierarchical structure to display the elements of a given model within a navigation explorer. However, unlike in TouchCORE, none of the tools provide a method for defining which types of model elements are pertinent for display. In all cases, clicking on the name of a model within the explorer causes that model to be displayed on-screen. However, only in the case of Sparx does this type of navigation extend to model elements as it does in TouchCORE. In two of the tools, StarUML and MagicDraw, there are some explicit links within the navigation system between inter-related models (such as between operations and their related Sequence Diagrams). However, in no case does this generalize in the way it does for Perspectives in TouchCORE. Finally, in none of the tools is a back button provided to return to prior

models.

Based on this survey, the TouchCORE navigation system compares well to navigation systems within these four popular software modeling tools, both open source and commercial. TouchCORE is able to show a hierarchical structure for elements in a given model and easily navigate between models by clicking on their name within the navigation bar in a similar way to the tools surveyed.

In addition, we believe TouchCORE provides valuable functionality beyond the tools surveyed in allowing for easy navigation between models which are related to each other via a Perspective. In addition, TouchCORE allows a language designer to define which types of elements are valuable to display in the navigation bar, thus reducing clutter for the user. Finally, the back button provided in TouchCORE allows the user to quickly return to previous models they have been working on.

6.2 Summary

In this chapter, we outlined a survey we performed of popular modeling tools to compare the newly implemented TouchCORE navigation system to similar systems. We find that many of the features in TouchCORE bear similarities to those created in other systems. In addition, we find that our implementation is able to provide added functionality for users, largely through its relation to the Perspective system in TouchCORE. In the next chapter, we summarize our work and introduce some opportunities for future improvements to the

navigation system.

Chapter 7

Conclusion

7.1 Overview and Contributions

In this thesis, we set out to implement a system to allow users to generically navigate through models in the growing number of languages available within TouchCORE. We also expand this system to be able to handle multiple models which are interlinked through the concept of a Perspective which defines connections between different languages to aid users in maintaining consistency between models which are describing the same system.

To facilitate this improvement of the navigation system, we update the TouchCORE metamodel to navigation concepts. Specifically, we introduce the concept of a navigation mapping which allows language designers to define which types of elements are important within a given language and therefore should be displayed within a navigation bar. These

navigation mappings are then automatically displayed within the navigation bar to illuminate to the user the pertinent elements that exist within the current model environment.

We define two different types of navigation mapping - Intra-Language and Inter-Language. Intra-Language mappings display the elements which exist within the current model and allow the user to highlight where they are located and see how they are interconnected. Inter-Language mappings show the user how elements derived from different models are related and allow a user to navigate between models to see other models which are modeling the same system and elements which are modeling the same concept in different models.

We also integrate these navigation concepts into several of the modeling concepts which make up the backbone of TouchCORE. We extend the navigation functionality to Feature and Impact models in order to streamline navigation within a Concern and make it clearer for the user. In addition, we define a specific feature of the navigation tool for instances of model reuse within TouchCORE. Navigation for Feature and goal models is built to utilize the same concepts as general languages but are coded to be available no matter the language or Perspective with which a user is currently modeling.

Finally, we present a set of tests which were used to validate the quality of the navigation tool. These tests were performed manually by the user to ensure that all of the features are able to work in tandem.

7.2 Future Work

We now overview some potential opportunities for work building on the implementation detailed in this thesis.

7.2.1 Filtering

As noted in Section 4.1.1, we have removed the proposed filtering feature from the current iteration of the navigation system. However, we believe this functionality would be beneficial for future iterations of the tool. The filtering system would be used in cases where a large number of elements are present in a given model and a user may wish to not see all of the elements of a given type within the navigation bar. For instance, in the case that a class diagram contains a large number of classes and operations, a user may only want to see the operations of a given class if they are public. The filtering system would allow that user to filter out all private and protected operations, making it easier to see only those which are pertinent to them at the time. This would require updates to the metamodel to include the filtering metaclasses which were removed during the evolution of the metamodel.

7.2.2 Graphical User Interface for Defining Navigation Mappings

As noted in Chapter 4, we intend to update the method by which a user defines the mappings for a new language or Perspective. The current system relies on a user encoding their choices for mappings within code using the metamodel for the navigation system. However,

this requires a somewhat considerable level of understanding of how the navigation system works. We wish to implement a graphical user interface (GUI) to allow a user to input the intended mappings for the navigation of a new language or Perspective without any knowledge of the implementation details of the navigation system. We intend to perform a study utilizing the methodology of Human-Computer Interaction (HCI) to find a system which provides a streamlined method of definition for existing TouchCORE users as well as those with no knowledge of the system details. This GUI may also be extended to include the definition of Perspective mappings as outlined in Section 2.2.

7.2.3 Automatic Testing of Navigation System

When testing the current version of the navigation bar, one must personally complete all of the testing situations outlined in Appendix A to ensure that any updates made to the system do not result in a failure of an already completed situation. This is, naturally, a very arduous process as the number of test cases is high due to large number of different situations and edge cases. An automated testing system for the navigation bar would remove a significant amount of stress on the developer by allowing them to be confident in any changes made not affecting the previous system. As this system is highly dependent on user inputs, a testing system would need to quantify the changes that a user could make to a given model and how the user interface should respond to those changes.

Bibliography

[1] H. Ali, G. Mussbacher, and J. Kienzle, "Action-driven consistency for modular multi-language systems with perspectives," in *Proceedings of the 12th System Analysis and Modelling Conference*, SAM '20, (New York, NY, USA), p. 95–104, Association for Computing Machinery, 2020.

[2] H. Ali, G. Mussbacher, and J. Kienzle, "Generic graphical navigation for modelling tools," in *System Analysis and Modeling. Languages, Methods, and Tools for Industry 4.0* (P. Fonseca i Casas, M.-R. Sancho, and E. Sherratt, eds.), (Cham), pp. 44–60, Springer International Publishing, 2019.

[3] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*, vol. 1. Synthesis Lectures on Software Engineering, 09 2012.

[4] "Staruml." `https://staruml.io/`.

[5] "Magicdraw." `https://www.nomagic.com/products/magicdraw`.

[6] B. Combemale, J. Deantoni, B. Baudry, R. France, J.-M. Jézéquel, and J. Gray, "Globalizing modeling languages," *Computer*, vol. 47, pp. 68–71, 06 2014.

[7] M. Schöttle, N. Thimmegowda, O. Alam, J. Kienzle, and G. Mussbacher, "Feature modelling and traceability for concern-driven software development with touchcore," in *Companion Proceedings of the 14th International Conference on Modularity*, MODULARITY Companion 2015, (New York, NY, USA), p. 11–14, Association for Computing Machinery, 2015.

[8] H. Ali, G. Mussbacher, and J. Kienzle, "Towards modular combination and reuse of languages with perspectives," in *22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pp. 387–394, 2019 ACM/IEEE, 09 2019.

[9] O. Alam, J. Kienzle, and G. Mussbacher, "Concern-oriented software design," in *16th International Conference, MODELS 2013*, (Miami, FL, USA), 2013 ACM/IEEE, 10 2013.

[10] K. Pohl, G. Böckle, and F. Linden, *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer-Verlag Berlin Heidelberg, 01 2005.

[11] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-oriented domain analysis (foda) feasibility study," Tech. Rep. CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[12] A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley Publishing, 1st ed., 2009.

[13] C. W. Krueger, "Software reuse," *ACM Comput. Surv.*, vol. 24, p. 131–183, June 1992.

[14] M. B. Duran and G. Mussbacher, "Top-down evaluation of reusable goal models," in *New Opportunities for Software Reuse* (R. Capilla, B. Gallina, and C. Cetina, eds.), (Cham), pp. 76–92, Springer International Publishing, 2018.

[15] S. Aljahdali, J. Bano, and N. Hundewale, "Goal oriented requirements engineering - a review," *Proceedings of the ISCA 24th International Conference on Computer Applications in Industry and Engineering, CAINE 2011*, 01 2011.

[16] W. Al Abed, V. Bonnet, M. Schöttle, E. Yildirim, O. Alam, and J. Kienzle, "Touchram: A multitouch-enabled tool for aspect-oriented software design," in *Software Language Engineering* (K. Czarnecki and G. Hedin, eds.), (Berlin, Heidelberg), pp. 275–285, Springer Berlin Heidelberg, 2013.

[17] J. Kienzle, W. Abed, F. Fleurey, J.-M. Jézéquel, and J. Klein, "Aspect-oriented design with reusable aspect models," *T. Aspect-Oriented Software Development*, vol. 7, pp. 272–320, 01 2010.

[18] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.

[19] "Visual paradigm." `https://www.visual-paradigm.com/`.

[20] "Sparx enterprise architect." `https://sparxsystems.com/`.

Appendix A

List of Tests

Table A.1: A list of the tests performed to validate the implementation of the navigation system.

#	**Previous Situation**	**Change Made**	**Expected Outcome**
1	Home Screen	Create new Concern	Change to Concern Edit Scene. Navbar should have single Feature
2	Feature Model with at least one Feature	Create new Feature	Navbar should automatically add new feature under list of features

Table A.1

3	Feature Model with at least two features	Add a requires/excludes between features	Navbar should keep same number of features but the one for which the requirement/exclusion is related should have a new submenu showing the requirement or exclusion
4	Feature Model with at least one feature	Delete a Feature	Feature should be removed from the navigation bar submenu
5	Feature Model with at least one feature	Rename a Feature in model	Name of feature should change to reflect the editing of the feature within the model
6	Feature Model with at least one feature	Rename a Feature in navbar	Name of the feature in the model to reflect the change within the navbar
7	Feature Model with at least one feature. Feature should have no Artifacts.	Create a new Artifact	Should navigate to the new artifact. Upon return to Feature Model, the feature for which the realization model was created should have a new submenu listing the new artifact

Table A.1

8	Feature Model with at least one feature. Feature should have at least one other Artifact	Create a new Artifact	Should navigate to the new artifact Upon return to Feature Model, the feature for which the realization model was created should have a new artifact in the artifacts submenu
9	Feature Model with no goals	Create a new Goal	New submenu should be added to the nav bar for impact nodes
10	Feature Model with at least one Goal	Add a new Goal	A new Goal should be added to the list under the already created submenu
11	Feature Model with at least one artifact.	Navigate into Artifact	New artifact section should be added to the navbar and there should be nothing inside the section when it is opened
12	In an Artifact model.	Click the back button.	Should return to the feature model and remove the model and feature sections from the navbar.

Table A.1

13	Artifact with one view in perspective with one language. No elements	Add a new element	Navbar should add a new submenu to the section and the element should be included there.
14	Artifact with one view in perspective with one language. At least one element.	Add a new element of the same type as is currently onscreen	Navbar should add the element to the already created submenu where the other element(s) of the same type are shown.
15	Artifact with one view in perspective with one language. At least one element.	Add a new element of a type that is currently not on screen	Navbar should add a new drop down submenu with the new element inside.
16	Artifact with one view in perspective with one language. At least one element.	Add a new element related to one of the previous elements which also is connected to an intra-language mapping	Navbar should add a new submenu for the existing element showing the new type of connection to that element.

Table A.1

17	In an Artifact Scene	Click on a feature within the concern navbar section	Return to the concern scene and light up that feature
18	In an Artifact Scene with at least one element	Click on the element in the navbar	Should light up the element within the model to show where it exists
19	In an Artifact model with at least one element. Only one of that type of element.	Delete that element from the navigation bar.	Should delete the submenu containing that element.
20	In an Artifact model with at least one element. More than one of that type of element.	Delete that element from the navigation bar.	Should delete that element from its submenu but submenu should remain.
21	In an Artifact model with at least one element. That element should have Intra-Languages associated with it.	Delete that element from the navigation bar.	Should delete the element as mentioned in previous two tests and should delete the submenus for the element itself.

Table A.1

22	In an Artifact Scene	Create a concern reuse	Should automatically populate the navbar with a new submenu containing that reuse. In addition, a new reuse should be created in the concern's navbar section as well.
23	In an Artifact Scene	Create a model extension	Should automatically populate the navbar with a new submenu containing that extension
24	In an Artifact Scene with at least one extension	Click on extension in nav bar	Should navigate to the extended artifact and should show clearly that we are now in a reuse context
25	In an extended Artifact Scene	Click on back button	Should return to the original scene and should no longer show that reuse is occurring

Table A.1

26	In an Artifact Scene with at least one model reuse	Click on reuse in nav bar	Should navigate to the reused model and populate the navbar with a section related to the concern for the reused model as well as one of the reused model. Finally, icon should clearly indicate that we are now in a reuse context. Under the icon should be a menu showing the previous model which is reusing this one.
27	In a reused Artifact Scene	Click on back button	Should return to the original Artifact Scene and repopulate the navbar with the exact same as before the navigation into the reuse occured. No icon should remain regarding reuse.

Table A.1

28	In a reused Artifact Scene	Create a second level reuse	Should automatically populate the navbar with a new submenu containing that reuse. In addition, a new reuse should be created in the concern's navbar section as well. Function should be the same as if in a normal artifact.
29	In a reused Artifact Scene with at least one model reuse	Navigate to second level of reuse	Should navigate to the reused model and populate the navbar with a section related to the concern for the reused model as well as one of the reused model. Should remove everything for the fornally reused model. Should add a second model to return to in the menu under the reuse icon

Table A.1

30	In a double reuse model.	Click the back button.	Should navigate back to previous model and repopulate the navigation bar with its concern and model sections. In addition, should remove a model from the reuse section so there is only one remaining.
31	In a double reuse model.	Click on the first reuse model within the reuse section.	Should navigate back to the first reuse model and repopulate the navigation bar with its concern and model sections. In addition, should remove a model from the reuse section so there is only one remaining.
32	In a double reuse model.	Click the original model within the reuse section	Should navigate back to the original model and repopulate its concern, feature and model sections. Should remove the reuse section from the navigation bar.

Table A.1

33	In a Feature Model	Create a new Artifact in a Perspective with at least two languages w/ perspective mappings between them.	Should navigate to one of the new Artifact models and should automatically create a feature and model section within the navbar. In the model section, there should be Inter-Language mappings connecting the models
34	In an Artifact model within a Perspective which has at least two languages	Click on the Inter-Language mapping for one of the other models.	Should navigate to the model and should repopulate the navbar with a new model section, removing the one for the old model. Feature section should be unchanged.
35	In an Artifact model within a Perspective which has at least two languages and at least one element in each of the models.	Create a new Perspective mapping between the two elements in the two different models.	Should add a new submenu below the name of the element in the current model which should show the name of the mapping. Should open to reveal the name of the other element.

Table A.1

36	In an Artifact model with at least one Perspective mapping between model elements in the current model and another model.	Click on the Inter-Language mapping for one of the elements in one of the other model(s).	Should navigate to the other model and highlight the model element in that model. The navbar section for the original model should be replaced with one for the new model.
37	In an Artifact model which has been navigated to via Inter-Language mapping.	Click on the back button.	Should return to the original model and should replace the model section in the navigation bar with the original model's.
38	In an Artifact model with at least one Perspective mapping between model elements in the current model and another model.	Delete the element which is used in one of the Perspective mappings.	Should delete both the element and thereby the Inter-Lanugage mapping from the navigation bar.
39	In a Feature Model with at least one goal	Click on the goal in the navigation bar	Should navigate to the goal model with that goal as a root. Should add a goal model section onto the navigation bar.

Table A.1

40	In a goal model.	Open the navbar section for the goal model	Should show the goals for the system as well as the features added to goal model.
41	In a goal model.	Add a new feature to the goal model.	Should add the feature to the elements menu in the navbar section for the goal model.
42	In a goal model.	Click on one of the other goals within the goal model section.	Should navigate to the goal model for which that other goal is the root.
43	In a goal model. Has been navigated to directly from the Feature Model.	Click on the back button.	Should navigate back to the Feature Model and remove the goal model section from the navbar.
44	In a goal model. Has been navigated to from another part of the goal model where a different goal is the root	Click on the back button.	Should navigate to the previous part of the goal model and should not remove the goal model section.

Table A.1

45	In a Feature Model with at least 2 features. At least one of which has an Artifact associated with it	Associate a conflict resolution model with a feature.	Should add the Artifact to the submenu for that feature in the navigation bar.
46	In a Feature Model with a conflict resolution associated (two features associated with the same Artifact).	Open the conflict resolution model with the original feature.	Should navigate to the Artifact model and add a Feature and model section to the navbar. Model section should be as normal but the feature section should have a drop down menu showing all of the other features that use this model to realize them.

Table A.1

47	In a Feature Model with a conflict resolution associated (two features associated with the same Artifact).	Open the conflict resolution model with the second feature associated with the model.	Should navigate to the Artifact model and add a Feature and model section to the navbar. Model section should be as normal but the feature section should have a drop down menu showing all of the other features that use this model to realize them. In addition, the feature section should be named after the feature that opened the model.
48	In a conflict resolution model.	Click on another feature that uses the model within the feature section.	Should change the model to be viewed as if it were inside the other feature. Feature section name should be changed to the name of the new feature and the section should now show the old feature in the drop down menu.

Table A.1

49	In a conflict resolution model having already switched features.	Click the back button.	Should inverse the changing of the feature. Feature section should be as it was originally.
50	In an Artifact for a language which can have multiple views.	Create another view within the model.	Should be shown within the navigation bar. However, there should be no submenu for that element even if there could be intra-language mappings associated with it.
51	In an Artifact for a language which can have multiple views. The navigation mappings of the views do not have the increase depth set to true.	Click on another view in the navigation bar.	Should navigate to the new view. Should replace the current view's section with the new section.

Table A.1

52	In an Artifact for a language which can have multiple views. The navigation mappings of the views have the field increase depth set to true.	Click on another view in the navigation bar.	Should navigate to the new view. Should add another section to the navbar with the new view's information,
53	In an Artifact for a language which can have multiple views having already navigated from a previous view.	Click the back button.	Should return to the previous view and not the previous model.

Appendix B

Navigation System Pseudo-Code

In this Appendix, we present the pseudo-code for the navigation system. Specifically, we outline the algorithms performed when a new model is called to be shown, a navigation bar section is opened, a user selects an element within the navigation bar, an element is changed within the current model, or the back button is selected.

Data: A model within TouchCORE selected by a modeler.
Result: The selected model is shown on screen along with relevant navigation bar sections.

1 Modeler calls for a new scene to be open;
2 The SceneCreationAndChangeFactory creates the scene if not previously open and calls for it to be displayed or displays the existing scene;
3 The scene is shown with all of the model elements relevant to it;
4 The NavigationBar is called to add a section pertaining to the scene;
5 The NavigationBarSection is pushed to the section stack and initialized with a NavigationBarMenu;
6 A NavigationBarNamer, characterised by the EObject of the model, is associated with the NavigationBarMenu;
7 A listener is created for the new navbar section;

Algorithm 1: Algorithm detailing navigation to a model.

Data: Modeler input selecting a section of the navigation bar be opened.
Result: The list of elements for a given model is shown within the navigation bar.

```
1  The namer is called to initialize the menu for the on screen model;
2  The namer calls the NavigationMappingHelper with the on screen model object to
     populate the navigation bar;
3  The NavigationMenuHelper creates a new IntraLanguageMappingTree which is used
     to facilitate the creation of a list of elements;
4  The NavigationMenuHelper retrieves the Navigation Mappings from the current
     Perspective or from the hard-coded information in the case of built in models
     (Feature Models and Goal Models);
5  The NavigationMappingHelper adds the model object to a list of possible source
     objects;
6  while there are possible source objects in the list do
7  |  The NavigationMappingHelper checks to see if any of the Navigation Mappings
   |    are relevant to the current object and if so adds them to the tree;
8  |  The destination of any Intra-Language Mapping which is added to the tree is
   |    added to a list of objects which can act as the source for mappings later in the
   |    tree;
9  |  A listener is made for the object to check if any information relevant to the
   |    navigation system is added, removed, or changed for this object;
10 end
11 The completed tree is returned to the NavigationBarNamer;
12 The namer calls for the creation of a new NavigationBarMenuElement for the tree;
13 for each child of the root node in the tree do
14 |  if the node has no children then
15 |  |  it is added to the list underneath its parent node within a drop down menu
   |  |    categorized by the type of mapping between itself and the parent;
16 |  end
17 |  else
18 |  |  it is added to the list in the same way but it also recurses back and begins
   |  |    the process as if it were the root of the tree;
19 |  end
20 |  the list is then sorted by name and each drop down menu is added in the same
   |    way
21 end
```

Algorithm 2: Algorithm detailing opening of navigation bar section.

```
Data: Modeler selected element within the navigation bar.
Result: The system responds by changing the on screen model or highlighting the
        relevant model element.
The system checks to see the element is related to an Intra or Inter-Language
 Mapping;
if it is an Intra-Language Mapping and not multi view then
    the model element is highlighted;
end
else if it is an Intra-Language Mapping and multi view then
    if the element is in the current view then
        it is highlighted;
    end
    else
        the view is changed by calling changeView in
         SceneCreationAndChangeFactory and a similar process to Algorithm 1
         occurs;
    end
end
else if it is an Inter-Language Mapping then
    if the element is a model then
        the scene is changed as in Algorithm 1;
    end
    else if the element is a model element then
        the scene is changed as in Algorithm 1 and the element is highlighted in the
         new scene;
    end
end
else if it is a reuse then
    the scene is changed in Algorithm 1 but a new Navigation Bar Section is added
     for the reuses and an element is pushed to the reuse stack;
end
else if it is a conflict resolution context switch then
    the scene is refreshed with a new navbar section for the new feature;
end
```

Algorithm 3: Algorithm detailing response to modeler selecting an element in navigation bar.

Data: Change made to the on screen model.
Result: The navigation bar section for the model is updated to reflect the change.

```
if a new element was added to the model then
    if the element is connected to other elements in the navigation bar by a
     navigation mapping then
        The element is added to the IntraLanguageMappingTree to reflect each
         navigation mapping it is a part of;
        The new element is added to the navigation bar in a similar way to the
         method shown in Algorithm 2;
        The list of elements is sorted to ensure ordering remains consistent;
    end
end
else if a element is deleted from the model then
    Each of the instances of the element is removed from the
     IntraLanguageMappingTree;
    The NavigationBarNamer for the current model is called and the element is
     removed from the list;
end
else if the name of an element is changed then
    The change of name is reflected automatically by the text listeners within
     NavigationBarNamer;
end
```

Algorithm 4: Algorithm detailing response to modeler changing an element within the current model.

```
Data: A modeler selects the back button
Result: The previous model is re-opened and the navigation bar is updated to
        reflect that model.
1  if the previous model is the feature model for a concern then
2  |  The model scene is changed as in Algorithm 1;
3  |  The navigation bar sections for the feature and realization model are removed;
4  end
5  else if the current scene is a reuse scene then
6  |  The model scene is changed as in Algorithm 1;
7  |  if the previous scene is a reuse scene then
8  |  |  The previous scene's model is removed from the reuse section;
9  |  end
10 |  else
11 |  |  The reuse section is removed from the model;
12 |  end
13 |  The concern, feature and realization model sections for the current model are
   |   removed;
14 |  The previous scene's concern, feature and realization model sections are added
   |   to the model as in Algorithm 1;
15 end
16 else if returning from a conflixt resolution context switch then
17 |  The feature section is updated to reflect the new feature;
18 end
19 else
20 |  The model scene is changed as in Algorithm 1;
21 |  The realization model section in the navigation bar is updated to reflect the new
   |   model scene;
22 end
```

Algorithm 5: Algorithm detailing response to modeler clicking on the back button.