FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

# Assignment of master's thesis

| | |
|---|---|
| **Title:** | Machine Actionable User Interface Description for Normalised Systems Code Expanders |
| **Student:** | Bc. Patrik Jantošovič |
| **Supervisor:** | doc. Ing. Robert Pergl, Ph.D. |
| **Study program:** | Informatics |
| **Branch / specialization:** | Managerial Informatics |
| **Department:** | Department of Software Engineering |
| **Validity:** | until the end of summer semester 2022/2023 |

## Instructions

The topic contributes to the joint research between FIT and the University of Antwerp. The assignment focuses on model-driven approach to UI generation, namely generating an ontology for UI description and its embedding in the expanders system through Normalised Systems Gateway Ontology for Conceptual Models (NSGO4CM).

1. Acquaint yourself with the Normalised Systems Theory, NSX Expanders and NSGO4CM.
2. Acquaint yourself with the IFML language for modelling user interfaces.
3. Analyse NS code expansion with respect to UI description and its code generation.
4. Design a metamodel of IFML as an OWL ontology for representing IFML models.
5. Design and implement a tool for transformation of IFML models into RDF/OWL models.
6. Design and implement a tool for mapping RDF/OWL IFML models with the NSGO4CM ontology.
7. Demonstrate your results on a case study.
8. Discuss your achievements and formulate conclusions.

*Electronically approved by Ing. David Buchtela, Ph.D. on 11 January 2022 in Prague.*

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master’s thesis

# Machine Actionable User Interface Description for Normalised Systems Code Expanders

***Bc. Patrik Jantošovič***

Department of Software Engineering
Supervisor: doc. Ing. Robert Pergl, Ph.D.

May 1, 2022

# Acknowledgements

First of all, I wish to express my gratitude to my supervisor, doc. Ing. Robert Pergl, Ph.D., for his valuable advice and guidance. I would also like to thank Ing. Marek Suchánek and Ing. Jan Slifka for their consultations and help throughout the work of this thesis. And last but not least, I would also like to thank my family and my girlfriend for their continuous limitless support throughout my studies.

# Declaration

In Prague on May 1, 2022 ………………….

## Citation of this thesis

Jantošovič, Patrik. *Machine Actionable User Interface Description for Normalised Systems Code Expanders.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

# Abstrakt

Modelovací jazyky pro uživatelské rozhraní, jako například Interaction Flow Modeling Language, poskytují nástroje pro všechny účastníky procesu vývoje softwaru, díky kterým mají možnost lépe komunikovat a navrhovat uživatelské rozhraní aplikace. Normalized Systems Theory, navíc popisuje jak takové aplikace vytvářet a upravovat tak, aby byly udržatelné a snadno rozšířitelné. To je možné pomoci definovaných princípů a struktur, které jsou generovatelné z elementů za pomoci expanderů kódu. V téhle diplomové práci jsou prozkoumané možnosti transformace modelů z Interaction Flow Modeling Language do elementů definovaných v Normalized Systems Theory a to za pomoci přechodné ontologie nazvané Normalized Systems Gateway Ontology for Conceptual Models.

**Klíčová slova** IFML, Interaction Flow Modeling Language, NS, Normalized Systems, OWL, Web Ontology Language, Transformace Modelů

# Abstract

User interface modeling languages such as Interaction Flow Modeling Language provide tools for participants of a software development process to better describe and communicate the design of the front-end of the application. Normalized Systems Theory, on the other hand, describes how to create this application in a sustainable way, resulting in an evolvable application. It does so by defining various principles and structures that can be generated from Normalized System elements using code expanders. In this thesis, the possibilities of transforming Interaction Flow Modeling Language models into Normalized System elements through Normalized Systems Gateway Ontology for Conceptual Models are explored.

**Keywords** IFML, Interaction Flow Modeling Language, NS, Normalized Systems, OWL, Web Ontology Language, Model Transformation

# Contents

# List of Figures

Chapter 1

# Introduction

## 1.1 Motivation

Conceptual models provide a way to describe, understand and represent complex systems. Its main goal is to represent the fundamental information, such as functionality and the basic structure of the system, in an understandable way for all interested parties. In the software development process, it typically provides a point of reference for system specifications and also serves as documentation of the system.

There are multiple modeling techniques and methodologies in the field of software engineering. Each of these models typically focuses on different parts and views of the system and is therefore suitable for different use cases. Typically, data models such as entity-relationship models or domain models are used to describe the underlying data.

However, designers or analysts might want to also focus on the representation of content, user interaction, and behavior of the front-end of software applications. This is where the discipline of User Interface Modelling and its languages such as Interaction Flow Modelling Language is commonly used. Interaction Flow Modelling Language allows the users to capture not only the structure of the graphical user interface but also an interaction between the system and the user.

Information systems are complex ever-changing organisms, as they are constantly reflecting the changes in the business processes and needs of the users. The more complex the software is, the more costly these changes become over the lifetime of the systems. These issues with maintainability and evolvability are addressed in the Normalized Systems Theory, which aim is to describe how to create software that is sustainable. It does so by defining theorems, principles, structures, and design patterns, that have been proven to eliminate the combinatorial effects of the changes. Furthermore, it provides tools for code generation from its own kind of conceptual model called Normalized Systems Elements.

In this thesis, we explore the possibilities of transformation from Interaction Flow Modelling Language models into Normalized Systems elements. This is done partly by the Normalized Systems Gateway Ontology for Conceptual Models, which is a gateway ontology into the world of Normalized Systems. Normalized Systems Gateway Ontology for Conceptual Models is currently being developed by Marek Suchánek as part of his dissertation thesis and is therefore used in its current unpublished format.

## 1.2 Objectives

This thesis aims to explore and analyze the possibilities of transforming the Interaction Flow Modeling Language model (IFML) into a Normalized Systems Theory (NS) usable format. To achieve this, the following partial objectives have been formulated and noted for later evaluation:

- **O1** - Research and document the problem domains, namely: IFML, Normalized Systems Theory, Normalized System Code Expanders, ongoing research into Normalized Systems Gateway Ontology for Conceptual Models, appropriate tools for IFML and ontology modeling, and available APIs for manipulating such models.
- **O2** - Design an ontology describing Interaction Flow Modeling Language metamodel and implement this ontology in the appropriate tool.
- **O3** - Design a transformation and modeling process for IFML model creation as ontology using formerly created metamodel ontology.
- **O4** - Implement automated transformation tool for IFML model transformation into ontology.
- **O5** - Design and implement mapping tool from IFML ontology models into Normalized Systems Gateway Ontology for Conceptual Models (NSGO4CM).
- **O6** - Demonstrate and evaluate the achieved results on a real-world inspired case study.

## 1.3 Methodology

In the first few chapters of this thesis, a theoretical background needed to understand the given problem domains is well researched and prepared.

In Chapter 2, the current state of Normalized Systems Theory, its theoretical fundamentals, and practical components such as code expanders and gateway ontology for conceptual models is documented.

In Chapter 3 and Chapter 4, the focus is on describing the IFML methodology in detail and UI modeling as a whole. As the main source for IFML research, the official ontology definition metamodel by Object Management Group (OMG) is used. Because the main objective is to implement and demonstrate IFML models' transformation to ontologies in the case study on the core concepts of the language, some parts of the IFML are deliberately omitted in this thesis.

In Chapter 5, the world of semantic web and ontologies is described shortly. Specifically, research on the Web Ontology Language (OWL), the Resource Description Framework (RDF), and available tools is prepared for later use in the implementation part of this work.

After the theoretical part of this thesis, the core part of this work is presented in IFML Transformation (Chapter 6). The IFML meta model ontology is designed, created, and tested, modeling against this ontology is demonstrated and explained on two proof-of-concept models. Furthermore, the automatic transformation application is implemented and once again validated and evaluated on our selected proof-of-concept models.

With the models transformed from IFML diagrams into ontology format, mapping from IFML meta model ontology into NSGO4CM ontology is proposed and prepared in a form required by the NSGO4CM transformation tool. Afterward, using prepared mapping, the IFML model in ontology format is once again transformed into the Normalized Systems.

In the Chapter 8 part of the thesis, a more complex IFML model is created based on a real-world use case and then automatically transformed, demonstrating the complex abilities of our solution.

Finally, in Evaluation (Chapter 9) and Conclusion (Chapter 10) the achieved results are summarized and evaluated. Additionally, the possibilities for future work are discussed.

Chapter 2

# Normalized Systems Theory

## 2.1 Introduction To Normalized Systems

In recent years, the software development process has become more agile, but the software itself stayed very unadaptable to changes. Normalized Systems theory was introduced as an approach to developing agile and evolvable software by defining theorems, principles, and design patterns. By adhering to these theorems and principles, evolvable architecture that is resistant to so-called `combinatorial effects` can be created.

## 2.2 Current State

Normalized System theory is being developed and commercially used at the University of Antwerp. Its goal is to provide a way to develop large-scale information systems that are evolvable. The concept of evolvability comes from the ever-changing business requirements of the information system. These requirements often lead to the increasing complexity of the information system and therefore the increasing cost of implementation.[1]

The foundation of NS theory comes from the system theory, mostly from the concept of system stability and also from the theoretical concept of entropy. In the theory, evolvability and stability are accomplished by the absence of `combinatorial effects`. These effects are defined as changes, whose complexity depends not only on the change itself but also on the size of the system.[1]

## 2.3 Theorems

Normalized Systems theory proposes four theorems/rules for system stability and evolvability. Adhering to these rules should, according to normalized systems theory, lead to disposing of `combinatorial effects`.[2]

In their very nature, these theorems are not revolutionary new ideas, but rather very well-known principles of software development. It is important to note, that following these principles is necessary not a sufficient condition for system stability.[1]

### 2.3.1 Separation Of Concerns

*Every change driver (concern) is separated from other concerns in its own module.[1]*
*A processing function can only contain a single task in order to achieve stability.[2]*

Separation of concerns is a commonly used practice in software development. It is a design principle that is used to divide the software into distinct sections. Each section addresses a separate concern, whereas a concern can be, for example, a tier in a multi-tier application or a specific functionality of the software.[2]

### 2.3.2 Data Version Transparency

*Data entities can be updated without impacting the entities using it as an input or producing it as an output.[1]*
*A data structure that is passed through the interface of a processing function needs to exhibit version transparency in order to achieve stability.[2]*

Data version transparency enforces the encapsulation of data fields. It is a principle that allows us to add and remove data fields in entities without affecting action entities using it.[2]
In object-oriented programming, this can be manifested on a higher level by hiding information at compile-time by using, for example, Java beans. On the lower level, the Java class with an empty constructor and getter/setter methods for data access can be used to achieve data version transparency.[2]

### 2.3.3 Action Version Transparency

*An action entity can be upgraded without impacting its calling components.[1]*
*A processing function that is called by another processing function needs to exhibit version transparency in order to achieve stability.[2]*

An action entity, by definition, should implement only one task. Therefore, there might be a need to change and/or replace the action. Action version transparency can be implemented using polymorphism, or by encapsulating the processing actions and using wrapper functions.[2]

### 2.3.4 Separation Of States

*Each step in a workflow is separated from the others in time by keeping state after every step.[1]*
*Calling a processing function within another processing function needs to exhibit state keeping in order to achieve stability.[2]*

During a workflow, the current state of actions is kept, which leads to stateful workflow systems. Storing the state, can for example be used when reacting to an error state caused by an exception. In this case, the action entity does not need to react to this exception by itself, but the error state is stored and a separate action entity can be used to react to it.[2]

## 2.4 Elements

Adhering to the introduced principles can prove to be very difficult, as every infringement of these theorems creates a `combinatorial effect`. Furthermore, following these principles can lead to a lot of boiler-plate code, such as wrapper classes and data accessors.

To represent these recurring constructs, five elements have been created. These elements are high-level design patterns that comply with the core principles of NS.[1][3]

### 2.4.1 Data Element

Data element is used to encapsulate and represent data variables and structures in an isolated module, adhering to the data version transparency principle.[3]

### 2.4.2 Task Element

The Task element is used to represent an instruction and/or function in an isolated module, adhering to the action version transparency principle. It can also be called an Action element.[1][3]

### 2.4.3 Workflow Element

A workflow element consists of multiple actions/tasks that are executed in a sequence. It can be described as a state diagram as the intermediate state is required for each task execution. Therefore state needs to be linked or be part of the data element that is the argument of the workflow execution.[3]

### 2.4.4 Connector Element

It is used as an interface, for users or external systems to interact with data elements and ensures that the action element is executed statefully. For each action element, a connector element can be derived.[3]

### 2.4.5 Trigger Element

The trigger element is used to verify states and decide if a Task should be executed. For each task element, a trigger element can be derived.[3]

## 2.5 Code Expanders

When the requirements are formulated in the form of NS elements, descriptor files are created. These descriptor files are XML/text-based files providing sufficient input for code expanders. The NS expanders expand these files into the source code and configuration, creating a working application in one of the supported technological stacks.[4]

This expanded code, while producing a working application, is often not enough and the application requires custom code to be added to the generated code. Traditionally, this would create problems for most of the code generators, however, in NS, such additions are expected. Custom code can be added inside a generated file in specific places marked by so-called `anchors`. These anchors are specified by the expanders themselves and are part of the code templates. Another option is to add a custom code as a whole external package that is later on called from the generated code.[4]

This custom code is `harvested` during regeneration in order to keep the customization intact. The procedure is pretty straightforward, it scans through the code in between the anchors and external packages, keeps them, and then reapplies them after the code generation.[4]

Chapter **3**

# UI Modelling & Design

In this chapter, we will briefly discuss the ideas behind modeling an application interface. The principles and description presented are largely influenced by the description from the IFML standard, as this is the path that we will follow for the rest of this thesis. But for now, we will try to avoid IFML-specific vocabulary as much as we can and stay strictly on the more abstract level of things.

When modeling internet or web applications we can start with thinking in terms of *containers*. The main window or main page template can be considered to be a top-level container, alternatively, we can have multiple page templates so we will have multiple top-level containers.[5]

Naturally, each container can be divided into multiple parts or sections and these containers can be divided into more and more sections. Therefore we can represent this structure as a *hierarchy of sub-containers* inside our top-level container.[5]

We can think of two examples, to demonstrate this. A classic desktop application, for example, IDE such as IntelliJ, and a classic web application, such as e-shop. In our IDE, we will have the main window (a top-level container), that can be divided into multiple parts (sub-containers) that are displayed simultaneously such as a top menu, a coding section, and a panel section. Our panel section can then be divided into more parts, for example, tabs, that are displayed exclusively. These are the attributes we also need to consider when modeling the application interface. On the other hand, in an e-shop, we might have multiple top-level containers for each page so we need to consider what is our default container that will represent our landing page.

Inside these containers, smaller units, called *components* can be found. Components can be used to display content (information) and/or enable the user to input the data.[5]

Naturally, when modeling UI, we also want to model the interaction that users can have with these *components* and *containers* and not just their mere existence. Consequently, we arrive at concepts such as *parameters*, used to transfer data from and to the components, and *events* describing what happens when interacting with the components. Furthermore, the effects, or results, of such events are represented in the form of *flows*, describing the change in the state of the user interface.[5]

To put this all into a simple example, let's say, we have a web application such as a mailbox. In our top-level *container*, we have a sub-container with a list of all the received emails. Our *component* can therefore be a representation of an email in this list. An *event* triggered by selecting one of these emails results in opening a new *container* (window/modal) with the message of the selected email. This may be represented by a *flow* connection between the *event* and the *container* that is produced to display the message, using the *parameter* as the identifier of the email to be presented.

## 3.1 Alternatives

As mentioned we will focus on IFML, but there are alternatives to IFML when it comes to the modeling user interface. Although, these are mostly in the field of web engineering. In this section, we will shortly go through the more notable ones.

### 3.1.1 UWE

UML-based web engineering is a software engineering approach for the web domain. Its main goal is to cover the whole life-cycle of the web development process. The focus of the UWE approach is to provide UML-based specific modeling language with security features, model-driven approach, and methodology, and also support for systematic design and automatic generation of web applications.[6]

The UWE notation is defined as a conservative lightweight extension of the Unified Modelling Language, lightweight meaning that a UML Profile is provided to satisfy the need of the web domain terminology and conservative in a sense that model elements of UML metamodel are not modified.[7]

### 3.1.2 WebML

Web Modelling Language's goal is to allow designers to model the features of a website on a level where detailed architecture is not needed. These features and concepts are represented by intuitive graphical elements, that should be readable by non-technical users and at the same time, should be provided in the XML representation that is machine actionable for code generators.[8]

It offers modeling perspectives for users to model. These are the Structural Model, Hypertext Model, Presentation Model, and Personalization Model. The structural model is used to model entities and relationships between them, such as the typical UML class model. The hypertext model is used to describe hypertext units and their navigation model. The presentation model expresses the layout and graphical representation of pages and content elements. And lastly, the personalization model is used to model users and user groups. These concepts allow us to describe the user-specific content of the web application.[8]

Chapter **4**

# Interaction Flow Modeling Language

This chapter provides a description of the Interaction Flow Modeling Language and its notation. It is divided into detailed subsections for each package (module). It should also provide sufficient information for the implementation part of this thesis, so the IFML metamodel can be modeled as ontology using this research part as documentation.

## 4.1 Introduction

The Interaction Flow Modeling Language provides a platform-independent description of a user interface for web, mobile and cross-platform applications. The goal is to provide a description of application structure and behavior as perceived by the user.[5]

With popular Model-View-Controller architecture in mind, the IFML focus naturally lies on the `view` part of the architecture. However, this can be extended on `controller` layer with regard to the events occurring between user and software. Furthermore, the `model` layer can be represented as data provided to the user.[5]

## 4.2 Modelling Aspects

Simply said, when modeling user interface, we need to address a few aspects that are pivotal to the UI design[5]:

- Structure of the view, in terms of independent, hierarchical, visual units displayed exclusively or simultaneously.
- Content of the view, in terms of data provided both by the application to the user and from the user to the application.

- Commands and inputs enabling user interaction resulting in events on the view.
- Reference to the actions triggered by user interactions and their results and effects on the state of the view after the action has been executed.
- Parameter passing information between displayed view and the actions that need to be performed.

IFML supports this by providing appropriate tools and components[5]:

- The view structure is defined using *View Containers*, their relationships, nesting to the hierarchy of view containers, and their attributes such as reachability, visibility, and more.
- The content of the view is modeled using *View Components* that are contained in view containers. IFML provides us with a wide range of components such as forms, details, lists, windows, and more but it can simply represent an image or an HTML input form.
- *Event* definitions, that might be initiated by the user, application, or an external system.
- *Event* transitions, describing how the state of the view is affected. This may be a change at the level of view components in the content displayed, an *Action* trigger, the change of the whole view container, or a combination of all.
- To model input-output dependencies between view containers or between a view container and an action, IFML offers *Parameter Bindings*.

## 4.3 Concepts

The IFML metamodel is divided into three packages, namely: `Core, Extension` and `DataTypes` package.[5]

In this section, we will provide documentation of the relevant parts of the metamodel, needed for our implementation part of this thesis. Some parts of the metamodel such as high-level abstract classes are omitted as they are not important with regard to our goals. Also, the `DataTypes` package is omitted as it is not necessary for our purpose.

### 4.3.1 Core Concepts

This package contains the concepts that are used to describe the infrastructure and interaction of the models.[5]

#### 4.3.1.1 View Element

While not listed as a core concept in IFML Standard, `View Element` is a core metaclass of IFML metamodel. It is an interface element that displays content. View elements are divided into two groups: `View Containers` and `View Components`.[5]

**Generalization**

`Interaction Flow Element`

#### 4.3.1.2 View Container

A `View Container` is an aggregating element that can contain other `View Containers` and/or interface elements displaying content.[5]

**Generalization**

`View Element`

**Attributes**

`isXOR` : Boolean

indicates that this container is mutually exclusive with his sibling containers

`isDefault` : Boolean

indicates that this container is displayed by default amongst his mutually exclusive sibling containers

`isLandmark` : Boolean

indicates global reachability of the container

#### 4.3.1.3 XOR View Container

A `View Container` with attribute *isXOR* set to *true* is `XOR View Container`. This means that all the included interface elements or `View Containers` are presented one at a time. Exactly one of the included objects is expected to have the *isDefault* attribute set to *true*, meaning that this particular child object will be presented to the user when the encapsulating `View Container` is accessed.[5]

This may be a container with multiple tabs, this functionality can be found in **bootstrap** open-source toolkit example.

#### 4.3.1.4 Landmark View Container

A `Landmark View Container` is a `View Container` with attribute *isLandmark* set to *true*. Therefore, becoming directly reachable from any `View Element` contained in the same `View Container`.[5]

This may be a Login/Logout button, or a Shopping cart link on websites that is visible on every page.

#### 4.3.1.5 Default View Container

As already mentioned in `XOR View Container`, this is the `View Container` that will be presented to the user when accessing its parent container.[5]

A welcome page of the website would be a `Default View Container`.

#### 4.3.1.6 View Component

It is an element of the interface that displays content or accepts an input. It can consist of multiple `View Component Parts`.[5]

This may be an HTML input form, an image, or other HTML elements displaying content on web site.

**Generalization**

`View Element`

#### 4.3.1.7 Event

An `Event` is an occurrence that can alter the application and its state. They can be **Throwing** or **Catching**, depending on the fact if they are thrown by the interaction or caught by it.[5]

**Generalization**

`Interaction Flow Element`

#### 4.3.1.8 Catching Event

`CatchingEvents` can cause navigation or parameter value passing between elements. They might be produced by user interaction, system notification, or by navigation.[5]

**Generalization**

`Event`

#### 4.3.1.9 Throwing Event

The event is induced by the modeled application when the triggering conditions are met.[5]

**Generalization**

`Event`

#### 4.3.1.10 System Event

This type of event is a catching event caused by a system, forcing the change in the user interface. This might be triggered by the fact that a certain time has elapsed, or by special conditions such as database connection loss.[5]

**Generalization**

`Catching Event`

**Attributes**

`type` : SystemEventType

indicates the type of system event

#### 4.3.1.11 View Element Event

It is a type of event representing a user interaction event triggered by `View Element`.[5]

**Generalization**

`Catching Event`

#### 4.3.1.12 Action Event

Type of event triggered by an Action.[5]

**Generalization**

`Catching Event`

#### 4.3.1.13 Action

Any business logic, either server-side or client-side, triggered by an `Event` is called `Action`. In addition, each action can trigger multiple `Catching Events` called *Action Events* as the result of business logic computation or exception occurence.[5]

**Generalization**

`Interaction Flow Element`

`Named Event`

#### 4.3.1.14 Navigation Flow

`Navigation Flow` represents navigation of `View Elements`, the Action processing or a System event. It can be accompanied by a set of `Parameters` that are passed to the target element through `Parameter binding`. Furthermore, corresponding `Data Flow` can also be triggered to pass further parameters to the target element.[5]

**Generalization**

`Interaction Flow`

#### 4.3.1.15 Data Flow

It is an element used to pass context information between various elements. It is triggered by `Navigation Flow` element to pass parameters to the target element.[5]

**Generalization**

`Interaction Flow`

#### 4.3.1.16 Parameter

Simply put, `Parameter` instance holds a value that is passed between elements. Parameter flow is executed when events are triggered. Parameters may belong to the elements of the user interface such as `View containers, View components, View component parts` determining their properties. Furthermore, parameters are restricted in their scope by the element that is holding them, meaning they can be accessed only by the elements from the same model space.[5]

**Generalization**

`Interaction Flow Model Element`

`Multiplicity Element`

`Typed Element`

`Named Element`

**Attributes**

`direction` : Direction

indicates the direction of the parameter, whether it is in or out, with the default being the input parameter

`defaultValue` : Expression

indicates the default value of the parameter, calculated through the expression

#### 4.3.1.17 Parameter Binding

To connect the input parameter of the target element and output parameter of the source element, basically defining how the parameter is passed, `Parameter Binding` is used. On `Event` trigger, this binding is followed, transferring the value from the source to the target element.[5]

**Generalization**

`Interaction Flow Model Element`

#### 4.3.1.18 Parameter Binding Group

`Parameter Binding Group` is an aggregation of the `Parameter Bindings` associated with an interaction flow.[5]

**Generalization**

`Interaction Flow Model Element`

#### 4.3.1.19 Activation Expression

It is a boolean expression associated with a `View Element`, `View Component Part` or `Event`, enabling the associated element when the expression is evaluated as true. It uses `Parameter` values for expression evaluation.[5]

**Generalization**

`Boolean Expression`

#### 4.3.1.20 Interaction Flow Expression

Boolean expression determining which of the interaction flows will be followed after an `Event` was triggered. It uses `Parameter` values for expression evaluation. When `Event` does not have an `Interaction Flow Expression`, all interaction flows associated with the `Event` are followed.[5]

**Generalization**

`Expression`

#### 4.3.1.21 Module

To improve maintainability, IFML offers `Module` creation. It is a reusable part of the user interface and corresponding actions. Optionally, it is associated with a set of `Ports`.[5]

**Generalization**

`Interaction Flow Model Element`

`Named Element`

#### 4.3.1.22 Port

Port is an interaction point between `Module` and the rest of the model. Its interface is defined by a `Port Definition`[5]

**Generalization**

`Interaction Flow Element`

#### 4.3.1.23 Port Definition

Port definition is used to define interface for the `Port`, holding `Parameters` for value passing to and from modules.[5]

**Generalization**

`Interaction Flow Element`

#### 4.3.1.24 Input Port

It is an interaction point between a `Module` and its surroundings. It collects all interaction flow and parameters arriving at `Module` and distributes them inside.[5]

#### 4.3.1.25 Output Port

It is an interaction point between a `Module` and its surroundings. It collects all interaction flow and parameters inside `Module` and distributes them to its surroundings.[5]

#### 4.3.1.26 View Component Part

It is an element that can only live inside the context of `View Component`. It may trigger `Events` and can have incoming and outgoing interaction flow. Activation of the `View Component Part` can be conditioned by the *activation expression* attribute. If this expression is not present, the element is considered active.[5]

`View Component Part` typically represents a field in a form.

**Generalization**

`Interaction Flow Element`

### 4.3.2 Extensibility

IFML provides a way to allow the definition of stereotypes, tagged values and constraints through standard UML extensibility mechanism. Extensions are supposed to provide a refinement or a specification of core concepts of the IFML. According to IFML documentation, following core concepts can be extended and still adhere to standard[5]:

- `ViewContainer`
- `ViewComponent`
- `ViewComponentPart`
- `Event`
- `DomainConcept`
- `FeatureConcept`
- `BehaviorConcept`
- `BehavioralFeatureConcept`

### 4.3.3 Extension Concepts

A set of extension concepts is provided by the IFML standard as an example for the extension mechanism.

#### 4.3.3.1 List

It is a type of `View Component` used to display a list of data binding instances. At least one instance must be associated with the list, i.e. the list cannot be empty. Furthermore, data binding instances can be associated with an `Event`, meaning that on each item selection, the `Event` is triggered.[5]

E.g. table with the content of the same type.

**Generalization**

`View Component`

#### 4.3.3.2 Form

It is a type of `View Component` representing an input form in which the user can provide information by filling out fields of the form. `Form` is required to have at least one field and typically a `Submit Event` is associated with the form.[5]

In HTML this is an equivalent of the *form* element.

**Generalization**

`View Component`

#### 4.3.3.3 Details

`Details` is a type of `View Component` used to display information from a data binding instance. Hence, it is mandatory for the `Details` component to have exactly one data binding instance associated with it. Furthermore, `Event` can be associated with component of this type, triggering the `Event` on display and passing `Parameter` values of data binding instance to the target interaction flow element.[5]

**Generalization**

`View Component`

#### 4.3.3.4 Menu

`Menu` is a special type of `View Container` used to model the concept of the menu that can be for example found on the web application as a navigation menu. The important constraint of this element is that it cannot contain any sub-containers or `View Components`.[5]

**Generalization**

`View Container`

#### 4.3.3.5 Field

A field is a pair of value-type that can be displayed to the users or can be used to capture input from the user. It also behaves as a parameter, passing values from and to elements.[5]

**Generalization**

`View Component Part`

`Parameter`

#### 4.3.3.6 Simple Field

`Simple Field` is a type of field that is used to display or capture value in a textual form.[5]

**Generalization**

`Field`

#### 4.3.3.7 Selection Field

This type of field enables the user to select one or more values from a predefined set of values.[5]

**Generalization**

`Field`

**Attributes**

`isMultiSelection` : Boolean

indicates whether elements allow selecting multiple values

#### 4.3.3.8 Validation Rule

`Validation Rule` is a kind of constraint that verifies the value supplied to the `Field` by the user.[5]

**Generalization**

`Constraint`

#### 4.3.3.9 Window

It is a type of `View Container` used to model the concept of the window.[5]

This can represent a new page in HTML or a new window in a desktop application.

**Generalization**

`View Container`

**Attributes**

`isNewWindow` : Boolean

indicates whether the container is opened in a new window

`isModal` : Boolean

indicates whether the window is rendered as a modal window

#### 4.3.3.10 Modal Window

`Modal Window` is a `Window` which when rendered, blocks interaction in previously displayed containers.[5]

#### 4.3.3.11 Modeless Window

`Modeless Window` is a `Window` that is placed over previously displayed containers that remain active.[5]

#### 4.3.3.12 Jumping Event

`Jumping Event` is a type of Throwing Event that triggers a navigation flow passing to referenced `Landing Event`.[5]

**Generalization**

`Throwing Event`

#### 4.3.3.13 Landing Event

References the destination of `Jumping Event`.[5]

**Generalization**

`Catching Event`

#### 4.3.3.14 On Select Event

`On Select Event` is a type of Event denoting the selection of an item from the multiple choices and passing its value to the target element.[5]

It can represent a selection of a row in a table or an item from the list.

**Generalization**

`View Element Event`

#### 4.3.3.15 On Submit Event

`On Submit Event` is a type of Event that triggers a parameter passing between interaction flow elements.[5]

In HTML this can be represented as a submit button of the form.

**Generalization**

`View Element Event`

#### 4.3.3.16 On Load Event

It is a type of `System Event` that is triggered when `View Element` that this event is attached to is completely rendered for the user.[5]

**Generalization**

`System Event`

## 4.4 Usage & Tools

A Systematic Literature Review has been performed in 2018 on the topic of Interaction Flow Modeling Language, which concluded that the IFML certainly simplifies the design and implementation of front-end interfaces, however, the existing tools are not mature enough to be utilized for complex and large software applications.[9]

In the following sub-sections, we will take a look at the most prominent modeling tools that support IFML.

### 4.4.1 IFML Editor Eclipse Plugin

IFML Editor is an open-source plugin based on Sirius technology with support for IFML standards in the Eclipse environment.[10]

Besides covering the whole IFML standard, being open source and multi-platform, other important aspects of IFML Editor are code generation-friendliness and extensibility to different domains based on Eclipse Modeling Framework. IFML Editor was created as a result of cooperation between WebRatio, research team AtlanMod, and Politecnico di Milano and it is available on GitHub or the Eclipse marketplace.[11]

### 4.4.2 IFML In Enterprise Architect

EA allows us to model IFML through the use of MDG Technology integrated with Enterprise Architect. According to the user guide, the following is provided for the user[12]:

- 11 IFML model patterns
- 2 IFML diagram types - model diagram and domain model diagram
- Core, Essential, and Extension concepts available in the toolbox

### 4.4.3 IFMLEdit.org

IFMLEdit.org is an online model-driven tool for the specification and rapid prototyping of mobile and web applications. It allows to model all important aspects of the IFML model[13]:

- The view structure and its content, i.e. `View Containers` and `View Components` along with their relationships, activation, and visibility attributes allowing for the display of content or data entry.

- The occurrences that affect the user interface can be caused by user interaction or by application in form of `Events`.

- The consequences of `Events` on the user interface in form of displayed content update, change of `View Container`, triggering off an `Action` or a mix of all of the above.

- The input-output dependencies between `View Elements` and `Actions` represented through `Parameter Bindings`

### 4.4.4 WebRatio

WebRatio is a model-driven, low-code development platform based on IFML. It consists of two parts. WebRatio Platform is focusing on web extended version of IFML while WebRatio Mobile Platform is implementing mobile-specific extensions for IFML. It provides three integrated environments[14]:

- **Modeling environment** supporting specification of IFML diagrams, UML class diagrams, and BPMN process diagrams

- **Development environment** to support implementation of custom components to allow personalized extensions of modelling language, custom functionalities, data and system integrations

- **Style design environment** and template layout for UI specification with support of HTML5, CSS and JavaScript

With user input combined from all environments, WebRatio is able to verify the model, generate code and manage the lifecycle of the product. Code generation results in cloud-deployed Java EE for web front-end and back-end with Apache Cordova for cross-platform mobile applications. More importantly, the generated code is human readable and maintainable without any closed components, therefore generated code can be managed even outside the WebRatio Platform.[14]

WebRatio might not be open-source, however, it is considered to be the richest tool from the one presented. Its code generation with specific platform support is unmatched amongst other tools and therefore it is a recommended tool for industrial purposes.[11]

Chapter 5

# Semantic Web

## 5.1 OWL 2

The W3C Web Ontology Language (OWL 2) is a semantic web language used to describe and represent knowledge and relations between things. It is used to express ontologies - which in our context means a set of precise statements about the domain of interests we are describing.[15]

It is a declarative language, not a programming one, describing the state of things in a logical way. Tools, so-called reasoners, can be used to deduce further information about ontology. There are multiple reasoners with various characteristics available, supporting different OWL2 profiles.[16]

As we are only concerned with OWL 2, which is an extension of the former OWL, we will interchange these abbreviations and use *OWL* and *OWL 2* as equals in the following sub-sections.

### 5.1.1 Basic Notions

To understand how knowledge is represented in OWL 2, some fundamental notions should be explained.

#### 5.1.1.1 Axioms

**Axioms** are the main components, the elementary pieces of OWL 2 ontology. It is a set of statements or propositions that defines what is true in the domain of interest. OWL 2 provides an extensive set of **Axioms**, for example: declarations and **axioms** about classes, data properties, object properties, data type definitions, keys, assertions and annotations.[17][15]

#### 5.1.1.2 Entities

**Entities** are elements, building blocks, used to describe and define the vocabulary of an ontology by referring to real-world objects. This includes classes, data types, object properties, data properties, annotation properties, and named individuals.[18][15]

#### 5.1.1.3 Expressions

**Expressions** are created by combining multiple names of **entities**. They are used to form complex descriptions from basic ones.[15]

### 5.1.2 Core

There are multiple syntaxes in the OWL standard. To describe various features in the following sub-sections, the *functional syntax* is used.

#### 5.1.2.1 Class

The most fundamental concepts in the domain correspond to the classes. They can be connected by using a transitive **SubclassOf** relation to create a hierarchy of classes, by relating a more specific class to the more general one. In OWL every user-defined class is implicitly a subclass of **owl:Thing**, therefore every instance is also an instance of **owl:Thing**. To describe a semantically equivalent class, OWL offers a transitive relation called **EquivalentClasses**. On the other hand, we might want to state that membership in one class excludes a membership in another. For this use case, relation **DisjointClasses** is used. To express more complex knowledge, intersection, union and complement of set theory is used, these are namely: **ObjectIntersectionOf, ObjectUnionOf, ObjectComplementOf**.[19][15]

#### 5.1.2.2 Individuals

A member of a class, more technically an instance of a class, is called an **Individual**. As in **classes** we can use relations to describe knowledge about these **individuals**. To describe inequality and equality of individuals we can use **DifferentIndividuals** and **SameIndividual** respectively.[19][15]

#### 5.1.2.3 Properties

**Properties** are binary relations used to describe facts between classes and individuals. We differentiate between two types of properties[19][15]:

- **Object properties** relate individuals to other individuals. We should note that the order of the individuals is important, i.e. motherOf, wifeOf, husbandOf relations.

- **Datatype properties** relate individuals to primitive data types like *string*, *int*, *dates*.

### 5.1.3 Open World Assumptions

It is important to note, and keep in mind while working with OWL 2, that it uses open-world assumptions. Traditionally, in closed-world assumptions, an unknown statement defaults to *false*, or more precisely, if it was not stated to be *true*, it is considered to be *false*. However, in an open world, we simply cannot consider it false, instead, we assume that this knowledge has not (yet) been added to the ontology.[20]

## 5.2 RDF

Resource Description Framework is an abstract modeling standard used to describe data by defining relationships between them. The basic data model consists of three types[21]:

- **Resources** - everything described by the RDF expression is a resource. They are identified by a URI and optionally by anchor IDs.
- **Properties** - are aspects, characteristics, attributes or relations used to describe a resource. Property has a meaning, set of permitted values, set of resources it can describe, and its relationships with other properties.
- **Statements** - a resource, property, and value of the property for a given resource together create a statement. These parts of the statement are called the subject, the predicate, and the object. The object, which is the value of the property can be another resource or a literal.

### 5.2.1 RDF/XML Syntax

As we established, the RDF is an abstract format that can be represented in different ways. These include Turtle syntax, N-Triples, RDF/JSON, and a few more.[22]

The most common one, however, is RDF/XML syntax which we are going to use in our thesis.[23]

## 5.3 SPARQL

It is a semantic query language able to retrieve and manipulate data stored in RDF format. Since RDF is essentially a directed labeled graph data format, SPARQL is naturally a graph-matching query language. It consists of three parts[24]:

- **Pattern matching** - includes features for graph pattern matching as optional parts, union, nesting, filtering values, and specifying a data source

- **Solution modifier** - used to modify values of an output produced by a computed pattern, modification can be made with typical operators such as projection, distinct, limit, or order
- **Output** - it can be of different types, new RDF data, selection of variables, a Yes/No query result

## 5.4 Protege

Currently, a go-to ontology constructing and maintaining the system (at least in the academic sphere) has come a long way from the 1980s when it was first introduced. With the emergence of OWL from the W3 consortium, Protege was at the time, the only ontology-development tool that could accommodate nearly the complete OWL specification. In the years to come, OWL2 emerged and Protege enhanced its support of the OWL2 standard becoming a highly popular tool used by more than 360 000 registered users as of now.[25][26]

Protege is a free, open-source ontology editor and framework for building intelligent systems. Being an open-source, extensible, Java-based system, it is supported by a strong community of academic, government, and business entities resulting in many plugins and add-ons to accommodate a wide range of use cases. The latest version fully supports the latest OWL 2 Web Ontology Language and RDF specifications from the World Wide Web Consortium.[26]

### 5.4.1 Protege API

ProtegeAPI is an open-source Java library for the Web Ontology Language. It can be used for both, developing plugins to the Protege application and creating standalone applications. The API provides an interface for loading, manipulating, and saving OWL files, querying, and modifying the ontologies. It also allows us to programmatically run reasoners on the loaded ontologies. It is currently maintained by Protege staff and the community.[27]

Chapter 6

# IFML Transformation

In this chapter, we will describe the transformation from IFML to RDF. Ontology will be constructed in `Protege` and a tool will be created to enable the transformation from IFML to RDF/XML that adheres to the created ontology. Modeling of IFML will be done in Enterprise Architect as this is the tool that the author of this thesis is the most familiar and comfortable with.

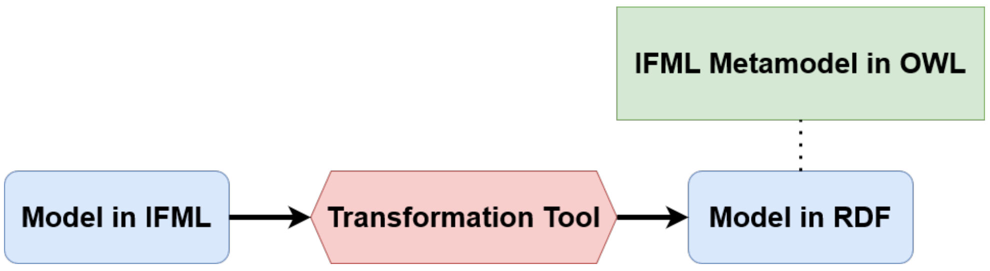


Figure 6.1: IFML Transformation Diagram

## 6.1 Ontology Creation

As a next logical step, we will now try to create a representation of IFML metamodel in `Protege`. We will try to create and adhere to simple rules for simple constructs and then try to use more advanced features to model more complex rules and constructs. This ontology will be created iteratively by continuously adding another layer of complexity to the ontology.

### 6.1.1 Creation Process

After some experimenting with the Protege ontology editor, and creating a few simple ontologies, the procedure has been proposed, to describe the modeling process of the IFML meta model ontology. This procedure also serves as a documentation of the set of rules used to design this ontology. The basic idea behind the transformation is to represent IFML classes as classes, attributes of the classes as data properties, and associations between these classes as object-properties.

This procedure is as follows:

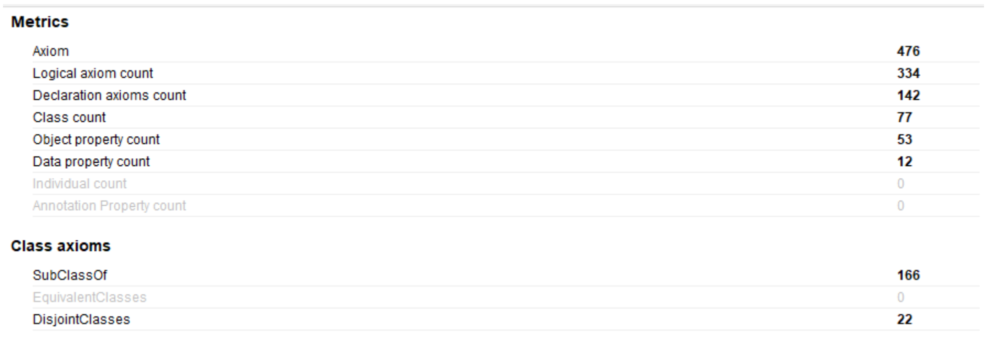
- For each `IFML class`, an `OWL class` is created.
- For each `generalization` relation in IFML metamodel, `subClassOf` relation is used in OWL.
- For each `attribute`, `data property` is created in ontology. The name of the data property is equal to the name of the attribute.
- For each `mandatory attribute`, data restriction property is created with cardinality *exactly 1*.
- For each `optional attribute`, data restriction property is created with cardinality *max 1*.
- Each `relation` with target or source outside of IFML metamodel is considered out-of-scope and is therefore ignored.
- For each `relation`, `object property` is created with `Domain` being the source of relation and `Range` being the target of the relation. Name of the object property is derived from prefix *has* and *Range or Relation name*, i.e. *hasDataBinding*.
- For each relation with the `same target`, existing object property is used and the `differing source` is added as an alternative in the `Domain` using *or* operator.
- Each `data property` is representing an attribute of IFML elements and is therefore marked as `Functional`.
- Each `class` with the same parent(s) is marked as disjoint with its siblings.
- For each `0..1 relation`, object restriction with `max 1` operator is created.
- For each `1..1 relation`, object restriction with `exactly 1` operator is created.

- For each `1..N relation`, object restriction with `some` operator is created.
- For each `0..N relation`, object restriction with `min 0` operator is created, bearing in mind that in open-world assumptions this is not very meaningful but we will still include it for completeness.
- `OWL class` siblings are marked as disjoint with `disjointWith` relation, otherwise, they might be considered equal by the reasoner.

Some exceptions have also been made as, for example, the proposed rules do not support creation of both *targetInteractionFlow* and *sourceInteractionFlow* relation on `InteractionFlow` class or *in/out InteractionFlows* on `InteractionFlowElement` class. This was resolved by indeed creating two `object properties` and specifying that they are `disjoint` to one another.

There was also an issue about handling *abstract* classes of IFML metamodel, as abstractness is not natural in OWL. It surely could be modeled as the fact that the abstract class is *unionOf* of these *disjoint* children's classes. However, in open-world assumptions, there is no type-checking present to enforce this logic, and we can simply have an instance of an abstract of which we do not know what type of sub-class it actually is. Furthermore, enforcing abstract classes might even pose a problem for us as not all implementations respect this constraint. For example, in Enterprise Architect implementation of IFML, `SimpleField` and `SelectionField` have been omitted and instead an abstract parent `Field` is used in models.

Following the procedure, we have arrived at our first prototype. The created ontology is depicted on fig. 6.3, using the *OntoGraf* plugin for Protege and as we can see from the metrics on fig. 6.2, we have arrived at 476 axioms in total.



**Metrics**

| | |
|---|---|
| Axiom | 476 |
| Logical axiom count | 334 |
| Declaration axioms count | 142 |
| Class count | 77 |
| Object property count | 53 |
| Data property count | 12 |
| Individual count | 0 |
| Annotation Property count | 0 |

**Class axioms**

| | |
|---|---|
| SubClassOf | 166 |
| EquivalentClasses | 0 |
| DisjointClasses | 22 |

Figure 6.2: IFML Ontology Metrics Attempt #1

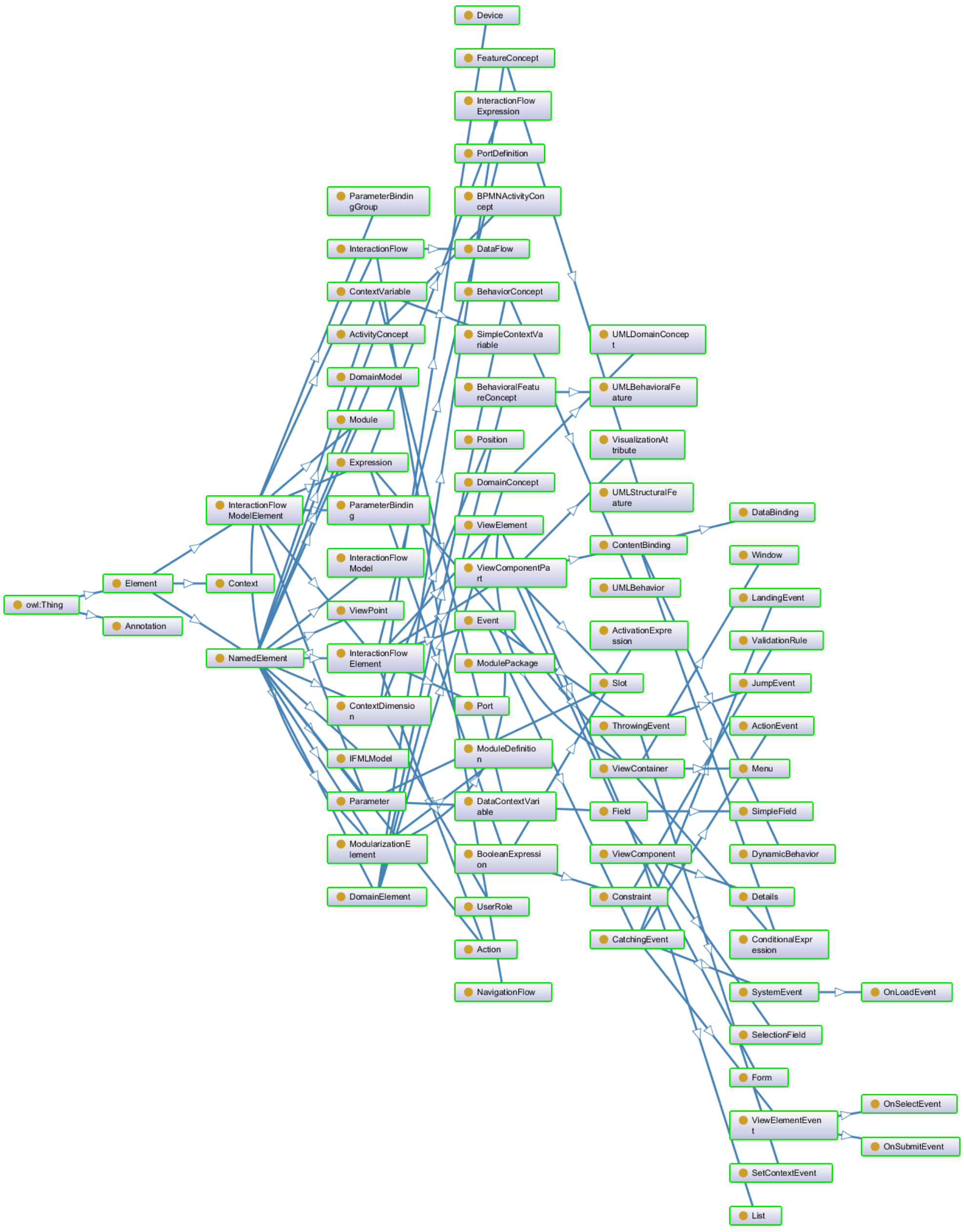


Figure 6.3: Ontograf Visualization of IFML Ontology Attempt #1

### 6.1.2 Testing & Refactoring

After the creation, HermiT reasoner has been used to check for any inconsistencies and incoherencies. The result was sub-optimal with many problems reported. Using Protege Debugger we have been able to identify the causes. Therefore, there have been some simplifications made.

Many of the problems resulted from having elements with multiple parents (generalization relation). After some investigation, we have arrived at the conclusion that there is no point in having `InteractionFlowModelElement` in our ontology as it only creates ambiguity and does not offer any additional information.

Another important point is that, once we want to start representing our IFML models in the ontology, we will use *NamedIndividuals* of Protege API. Therefore, we will expect every modeled element to be an `NamedElement`. This has been resolved by merging `NamedElement` and `InteractionFlowModelElement` into `NamedElement` drastically reducing inconsistencies. However, we will also respect the IFML metamodel in the sense, that every instance of `NamedElement` will also have an attribute (an OWL data-property) **name** set to the same value as is the *name of the NamedIndividual.*

Furthermore, every `Element` of IFML, is expected to have an id attribute, which once again is represented as a data property in OWL. Normally, this would be a simple identifier of some sort, its format is up to the author. However, this is convenient as we can use this further down the road to keep a relation between the IFML element to the automatically transformed one.

After the simplification we have done and running the Debugger with HermiT reasoner, we now get the message that our ontology is **coherent** and **consistent**.

### 6.1.3 Final Product

After following the proposed procedure and some simplifications, we have arrived at the prototype that is sufficient for our use case. There definitely is a lot more that can be done, in terms of defining constraints, fine-tuning the definition of relations, implementing the DataTypes package of IFML, and many more improvements. However, as we are going to illustrate in our proofs-of-concept, this prototype is a solid base for our research.

Once more, we depicted our ontology on fig. 6.4, this time connecting elements only by *subClass* relation for simplicity, using the *OntoGraf* plugin for Protege. And as we can see from the metrics on fig. 6.5, axiom count has been slightly decreased to 456.

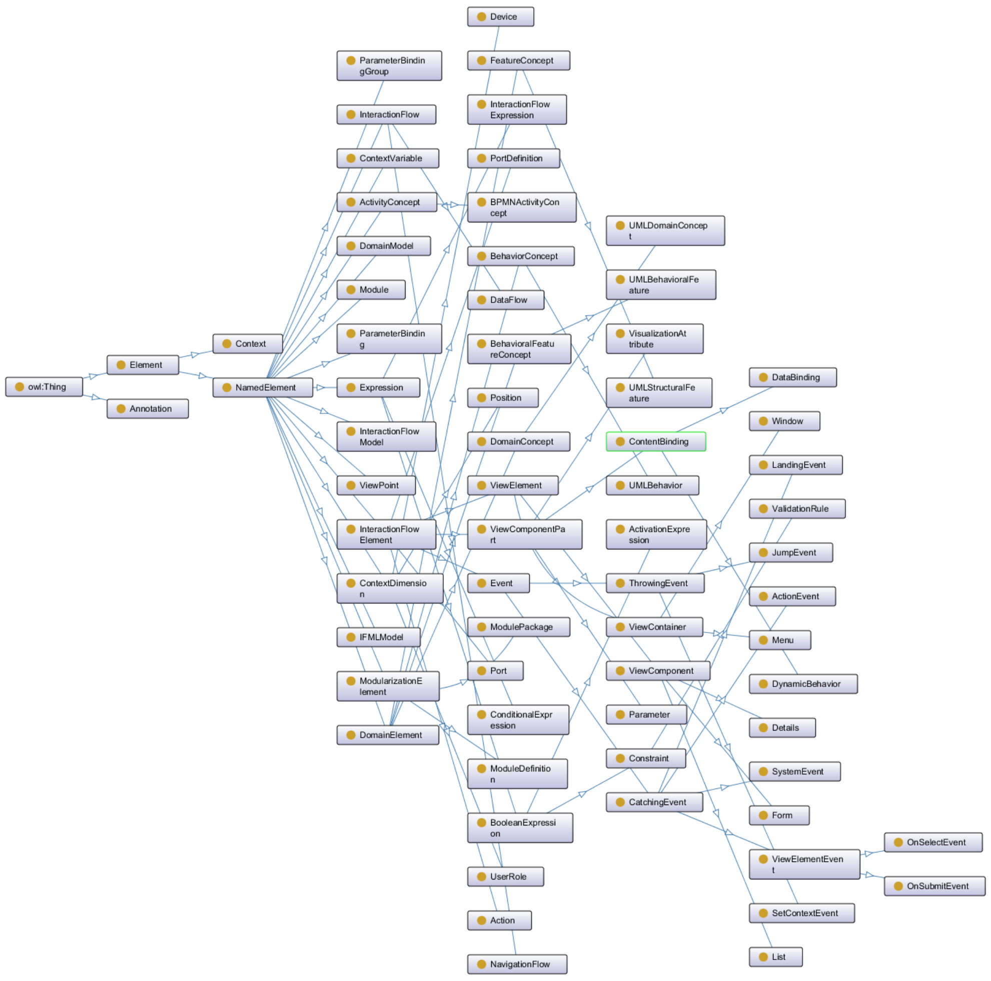
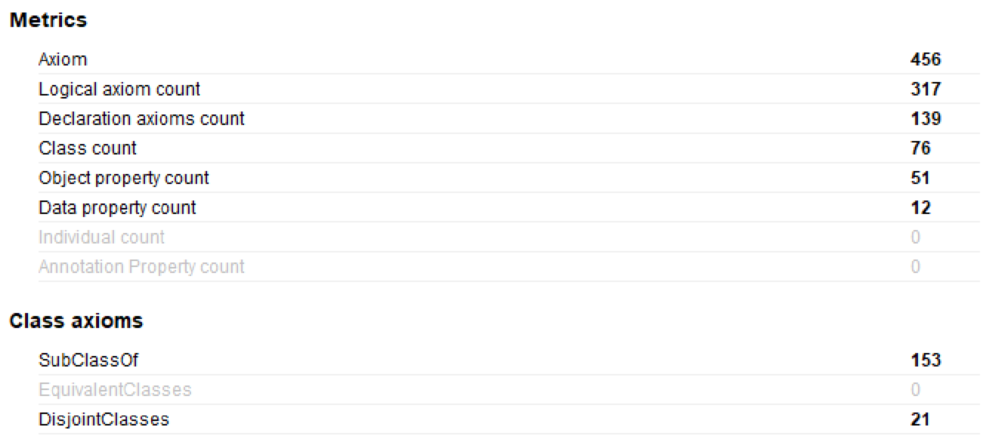


Figure 6.4: Final visualization of IFML ontology



**Metrics**

| | |
|---|---|
| Axiom | 456 |
| Logical axiom count | 317 |
| Declaration axioms count | 139 |
| Class count | 76 |
| Object property count | 51 |
| Data property count | 12 |
| Individual count | 0 |
| Annotation Property count | 0 |

**Class axioms**

| | |
|---|---|
| SubClassOf | 153 |
| EquivalentClasses | 0 |
| DisjointClasses | 21 |

Figure 6.5: Final IFML Ontology Metrics

The final ontology can be found on GitHub:
`https://github.com/PatrikJantosovic/ifml-ontology`.
The version tagged with **0.0.1-RELEASE** tag is used in demonstrations in the following chapters. In case there is more work done, for example, support of transformation into normalized systems, the application will be versioned accordingly.

## 6.2 Proof Of Concept

At the start, we will select two simple examples that will be used as proof of concept. First, we will model these diagrams in IFML, after that we will try to recreate the same diagrams in RDF in our created ontology. Using the knowledge of the source and target state, we will try to generalize transformation rules and implement them. The resulting tool will be an open-source Java application published on Github.

### 6.2.1 IFML

Examples are going to be simplified parts of the more comprehensive case study that can be found in the last chapter. These examples will later be used as input for our transformation application. In this section, we will try to demonstrate that we are able to work with different sets of elements, their attributes, and the relationships between these elements. We are using Enterprise Architect 14.1 and its built-in support for IFML modeling that is based on the OMG's UML profile for IFML using MDG technology. We will create a project, that will contain both of the proof-of-concept examples in separate packages and this project will be part of the submitted source code of this thesis.

#### 6.2.1.1 Example #1

The first example will be a simple homepage model with a list of categories, recommended products section, a shopping cart, and the main menu representing the basic navigation menu of the web application as you can see on fig. 6.6.

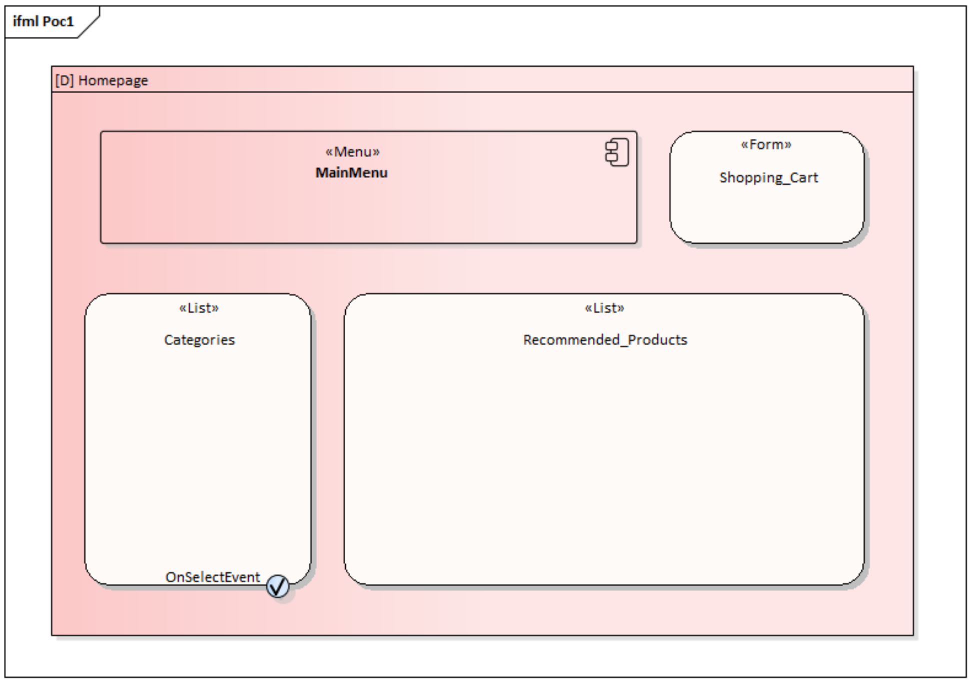


Figure 6.6: IFML Proof Of Concept Example #1

The goal here is to demonstrate support for basic IFML elements and the ability to create simple associations by nesting the elements, using different types of parent-child relationships.

In this example, we can already see that we will have to keep inheritance (through generalization association) in mind when transforming these associations. It is because, we can nest basically any element into each other and therefore create structure, however in IFML metamodel there is a relationship defined somewhere on the parent level of these elements, depending on the relationship. Therefore we also have them represented in our OWL metamodel as object properties between potential *superclasses*. This trait, therefore, has to be reflected in our transformation application.

#### 6.2.1.2 Example #2

And our second example will be a simplified checkout process as displayed on fig. 6.7.

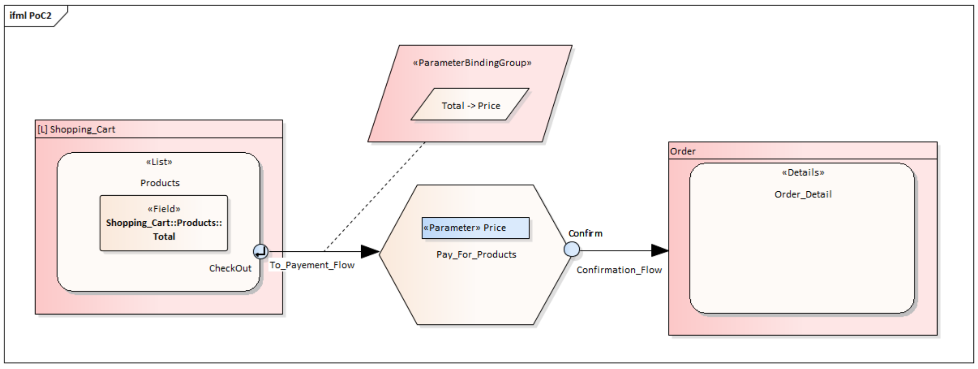


Figure 6.7: IFML Proof Of Concept Example #2

Here, the goal is to demonstrate the ability to create simple processes, using flows and bindings as relationships, which is semantically a different type of association to the one demonstrated in proof-of-concept example #1.

Both of these proof-of-concept figures can also be found as external attachments of this thesis, for better readability, identified as **I1** and **I2** respectively.

#### 6.2.1.3 Issue With ParameterBindings

However, looking at the diagram on fig. 6.7, it might be easy for humans to recognize that *PriceBinding* means passing a value from *field Total* to the *parameter Price*. And indeed this is how parameter bindings are drawn, as we can see on fig. 6.8, which is an example from the official IFML website.[28]

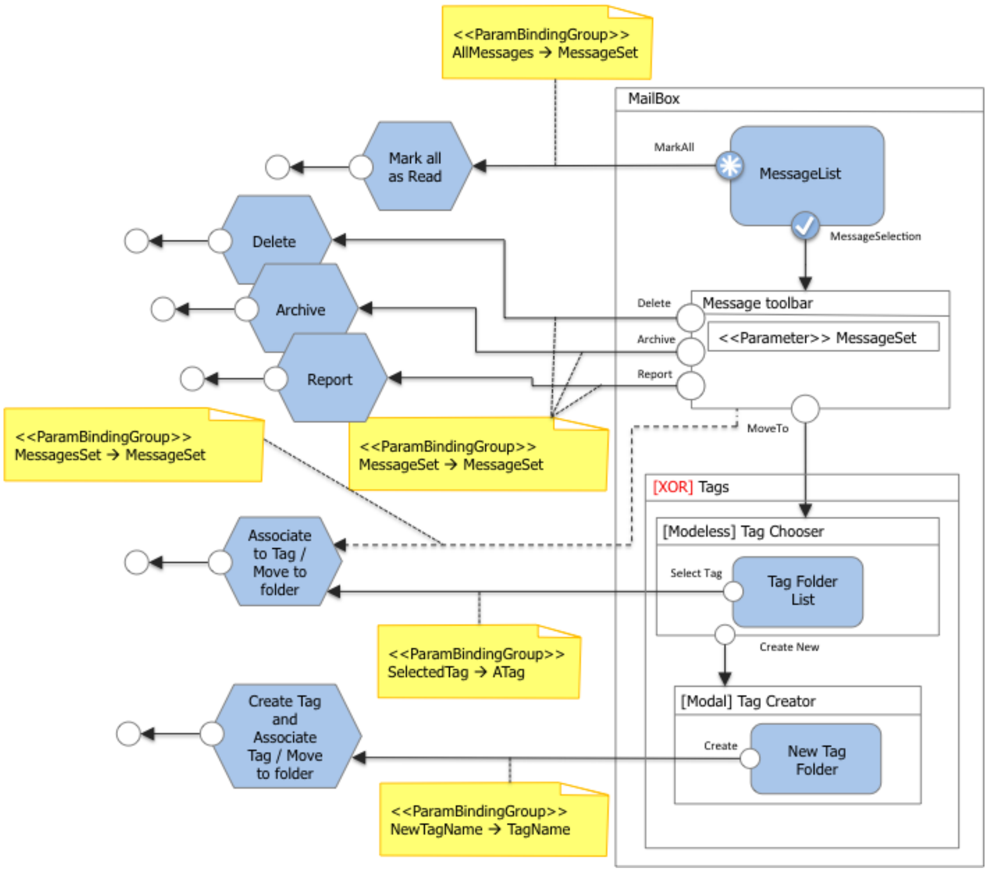


Figure 6.8: IFML Gmail Example [28]

The problem with this is that this is not machine-actionable. There is no way for a machine to understand which parameter binding corresponds to which parameters if there is no real and tangible relationship. Referring to the IFML documentation, we can clearly see that there should be two associations on `ParameterBinding` element, having both the source parameter and target parameter of the binding. However, this is not the case in the Enterprise Architect implementation of IFML.

Furthermore, there is another issue of the same type, with connecting `NavigationFlow` to the `ParameterBindingGroup`. There is a *Parameter Binding Link* association in the Enterprise Architect implementation, however, this is not part of the IFML standard and it is merely a *NoteLink* which, unfortunately, is not even exported when trying to save the diagram in XMI normative format. Rendering this link is unusable in our automatized transformation.

There are multiple ways of resolving this problem:

- We can come up with special naming rules for these elements and parse these names for additional information during transformation. This might be the least intrusive solution but it also is not very exact.
- Another possible solution is to use the classic **association** relationship, connecting source and target parameters to the parameter binding element and replacing the note-link relationship when connecting the parameter binding group to the navigation flow. However, this makes the diagram less readable and creates unnecessary complications once exported to the XMI file.
- Another option is to use **tagged values** to add properties to these elements as needed. The name of the tagged value will be the name of the *object property* of the corresponding relationship and the value will be the name of the related element. For example, in example #2, we will create a tagged value with the name *hasSourceInteractionFlow* on the `ParameterBinding` element with the value set to *Total*.

Every option requires some additional work from the user who wants to use our transformation application. Therefore, we will select the most simple one from both application implementation and modeler point of view, which is clearly the usage of the tagged values. Consequences on the structure of the XMI file and implementation details will be discussed in section 6.3 part of this work.

### 6.2.2 RDF

To model our proof of concept in RDF, we will once again use `Protege`. The results will be compared to the output of our transformation application and hopefully, the files produced would correspond to our modeled examples. In the beginning, a new project is created, and the existing IFML ontology is imported. In each example, we will create an `Individual` for every IFML element with appropriate characteristics and relations. We are free to use `NamedIndividuals` as we expect every IFML element to be a `Named Element`. This is a restriction (and a change in the ontology compared to IFML metamodel) that we have created to simplify modeling and transformation automation.

#### 6.2.2.1 Example #1

For our proof-of-concept #1 we will create the following `Individuals`:

- **Homepage** of type `ViewContainer`
- **Categories** of type `List`

- **Recommended_Products** of type `List`
- **Shopping_Cart** of type `Form`
- **MainMenu** of type `Menu`
- **CategorySelect** of type `OnSelectEvent`

As a next step, we should set `name` and `id data properties` for each created individual. Then, we will set the rest, which is fairly simple as we only have them on *the Homepage*. We will set them equivalently to the IFML values, that being `isDefault` is true and the rest of them are set to false.

Finally, we have to set appropriate relations using `object properties`. Looking at the diagram, we can clearly see the relations:

- *MainMenu* is a sub-container to the *Homepage*. In our implementation of IFML metamodel, this is simulated by using `hasViewContainer` object property which is an IFML equivalent of *view Container [0..1]* relation on `View Element` class.
- The *Categories, Recommended_Products*, and *Shopping_Cart* are all `View Components` belonging to the *Homepage*. Through generalization, these are also `View Elements`, we will once again use the `hasViewContainer` object property to describe the relation.
- Lastly, we have an `On Select Event` belonging to the `List`, which conveniently does have a `hasOnSelectEvenent` object property, which is the IFML equivalent of *On Select Event [0..*]* association on `List` class.

It is important to note, that there may be more `object properties` that could and/or should be set, but in this case, we can benefit from the open-world assumptions. It is not necessary to define all the values as far as our translated model provides the same set of information to the user, which in this case does.

The final product of our example #1 is therefore pretty simple as we can see on fig. 6.9. Id and name data properties have been omitted from the graph. The graph can be also found in attachments under **R1** identifier.

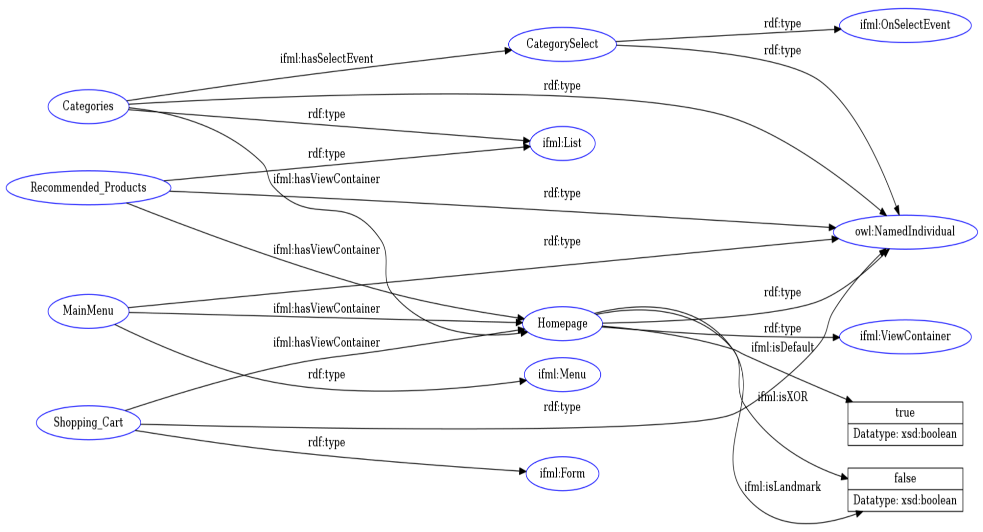


Figure 6.9: RDF Proof Of Concept Example #1

#### 6.2.2.2 Example #2

And for our second proof-of-concept example we will create the following `Individuals`:

- **Pay_for_Products** of type `Action`
- **Confirm** of type `ActionEvent`
- **Price** of type `Parameter`
- **Order** of type `ViewContainer`
- **Order_Detail** of type `Details`
- **PriceParameters** of type `ParameterBindingGroup`
- **Total-Price** of type `ParameterBinding`
- **Shopping_Cart** of type `ViewContainer`
- **Products** of type `List`
- **Total** of type `Field`
- **CheckOut** of type `OnSubmitEvent`
- **To_Payement_Flow** of type `NavigationFlow`
- **Confirmation_Flow** of type `NavigationFlow`

Then we naturally set `name` and `id` data properties, furthermore we set the `isLandmark` on shopping cart to true as we have done in IFML and the rest of the `ViewContainer` elements attribtues are set to false.

Now we have to find relations and their corresponding representation in our ontology. Once again, from the diagram, we can quickly identify the following:

- *Products* is a `List` belonging to the *Shopping Cart* element. Therefore, `hasViewContainer` object property will be used.
- *Order_Detail* is of a `Details` kind, belonging to the *Order* container. The `hasViewContainer` object property will be used.
- *Price* is a `Parameter` belonging to the *Pay_For_Products* action, therefore `hasParameter` object property will be used.
- *Total* is `Field` belonging to the *Products* list. Since it is a descendant of the `Parameter`, once again `hasParameter` object property can be used.
- *CheckOut* is a `OnSubmitEvent` belonging to the *Products* list, therefore `hasViewElement` object property can be used.
- *Confirm* is an `ActionEvent` that is related to the action with name *Pay_For_Products*. Therefore, `hasActionEvent` can be used.
- *Total-Price* is a parameter binding belonging to the parameter binding group. Therefore `hasParameterBinding` is used.
- *To_Payement_Flow* is an `NavigationFlow`, which requires us to specify the source and the target of the navigation. On the IFML diagram (fig. 6.7) we can clearly see that it starts in *CheckOut* and ends in *Pay_For_Products*. The `hasSourceInteractionFlowElement` can be set to the *CheckOut* and target of the relationship connected through `hasTargetInteractionFlowElement` to *Pay_For_Products* value.
- We will do exactly the same for the second instance of `NavigationFlow` that is present in our model. The `hasSourceInteractionFlowElement` to the *Confirm* and `hasTargetInteractionFlowElement` to *Order* value.
- Similarly, we need to provide both of the `hasSourceParameter` and `hasTargetParameter` object properties for the *Total-Price* parameter binding.
- Lastly, we have to connect the `ParameterBindingGroup` to the navigation flow *To_Payement_Flow* by using `hasParameterBindingGroup` object property.

The result is a more complicated RDF model than we had in example #1, as we can see on fig. 6.10. Once again, some elements and relationships were omitted for better readability. The full RDF diagram can be found in external attachments under the identifier **R2**.

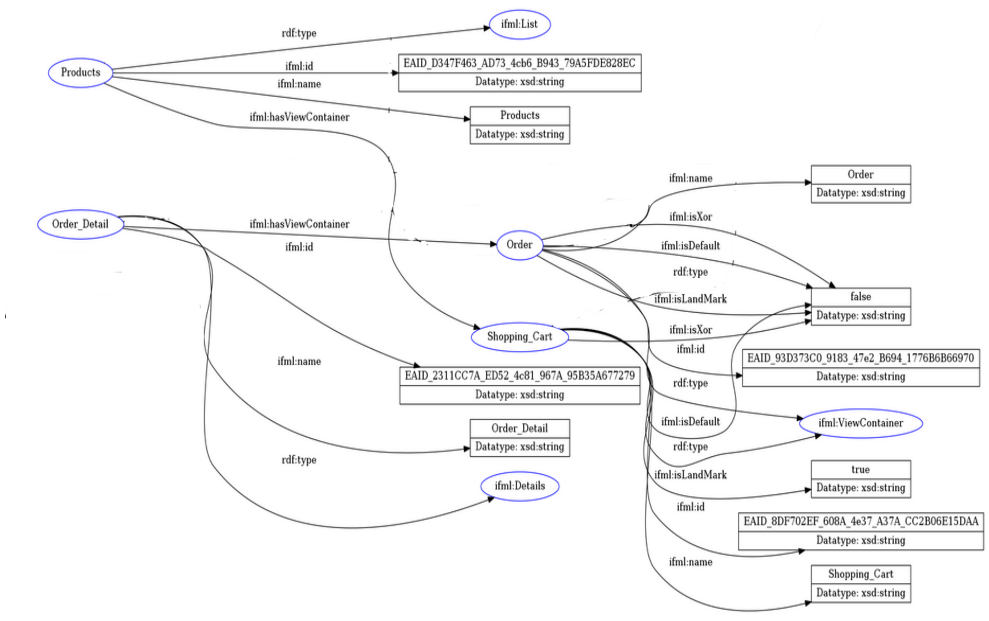


Figure 6.10: RDF Proof Of Concept Example #2 Snippet

## 6.3 Transformation Implementation

In this section, we will try to implement an application for the automated transformation of IFML models to their RDF/OWL representation. The goal is to create a simple **java** command-line application that works against our created IFML meta model ontology and is able to generate RDF/OWL files that adhere to this ontology.

To work with the ontology, we will use the already mentioned Protege API from section 5.4.1.

### 6.3.1 Input/Output Definition

First, we need to define our expected inputs and outputs. Trivially, our output is a file in RDF/OWL format representing the IFML model as ontology. Output ontology will be validated during the transformation against the metamodel.

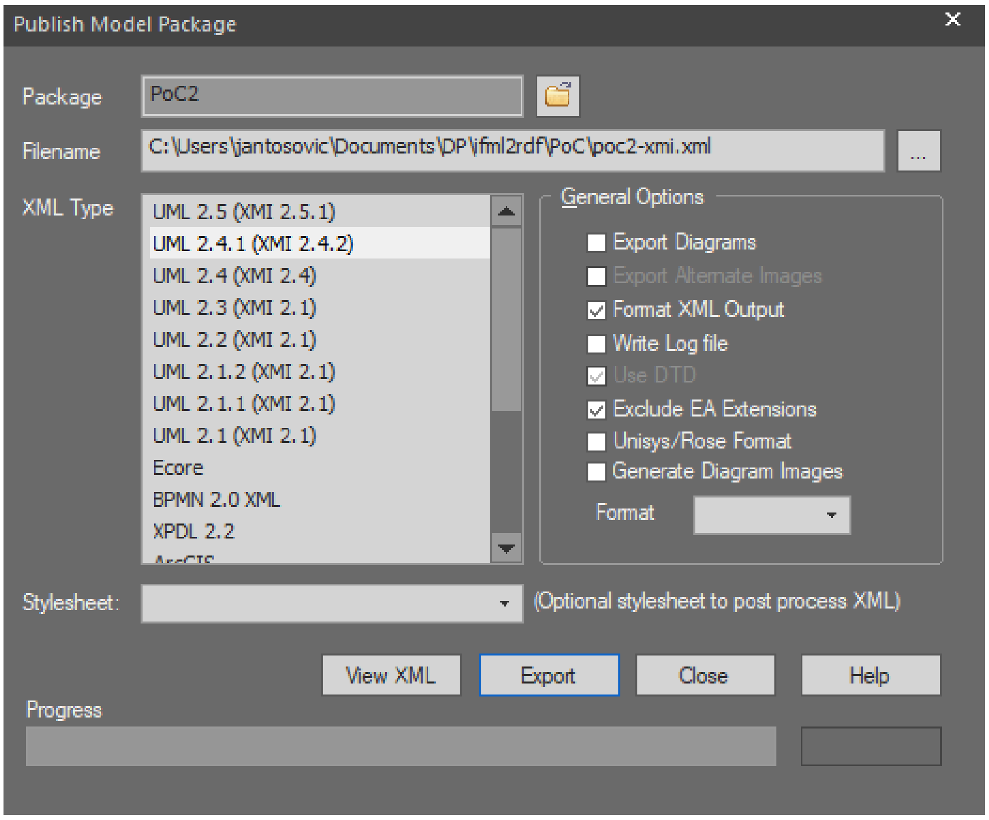
And for the input of the transformation application, we need to define the following:

- A path to the IFML meta model ontology in a machine-readable format for Protege API.
- An IFML meta model ontology base IRI to tell Protege API which ontology it should read from the file.
- A path to the destination where the resulting RDF/OWL file should be stored.
- An IRI of the target ontology.
- A path to the source file from which the IFML model should be read.

For the OWL part of the implementation, the ontology representation is pretty much set in stone, resulting from the need to adhere to Protege API. Obviously, the most common RDF/OWL format of the file is supported, for both loading and saving the ontology so there is no reason for us to go for a less standardized format.

However, we do need to find a suitable format for our IFML model implementation. Since we are using Enterprise Architect we do have multiple options, however, we are also restricted by IFML implementation in Enterprise Architect and its shortcomings as we have already mentioned in section 6.2.1.3. Looking at the options in Enterprise Architect, the XMI format is our choice as this is the format that is commonly used for the serialization of modeling languages.

Enterprise Architect offers multiple versions of XMI with various options. For our implementation, we want to keep the resulting XML file as simple as possible, yet we do need it to contain all of the necessary information. By a quick trial and error procedure and comparing multiple versions of the XMI file, we have arrived at the conclusion that the **Normative XMI 2.4.2** format will be used. That is equivalent to the following settings, as depicted on the fig. 6.11. Please note that we deliberately check the *Exclude EA Extensions* checkbox and uncheck the *Export Diagrams* checkbox as we would otherwise end up with unnecessary information in the result file.


Publish Model Package

Package: PoC2

Filename: C:\Users\jantosovic\Documents\DP\ifml2rdf\PoC\poc2-xmi.xml

XML Type:
- UML 2.5 (XMI 2.5.1)
- UML 2.4.1 (XMI 2.4.2) ✓
- UML 2.4 (XMI 2.4)
- UML 2.3 (XMI 2.1)
- UML 2.2 (XMI 2.1)
- UML 2.1.2 (XMI 2.1)
- UML 2.1.1 (XMI 2.1)
- UML 2.1 (XMI 2.1)
- Ecore
- BPMN 2.0 XML
- XPDL 2.2

General Options
- [ ] Export Diagrams
- [ ] Export Alternate Images
- [✓] Format XML Output
- [ ] Write Log file
- [✓] Use DTD
- [✓] Exclude EA Extensions
- [ ] Unisys/Rose Format
- [ ] Generate Diagram Images

Format:

Stylesheet: (Optional stylesheet to post process XML)

Figure 6.11: Enterprise Architect Publish Settings

#### 6.3.1.1 XMI File

Here we will shortly discuss the structure of our input XMI file and identify all the information that it provides and how we can process and use this information.

Looking at our example #1 representation on listing 1, we can take a note of the following five things:

- All IFML elements are listed at the end of the file, in *IFML:* namespace. This is very convenient, as we can easily identify which `Individuals` we need to create just by parsing elements with *IFML:* prefix from the file.

- As we established, we consider every element to be `Named Element` as we want to use `NamedIndividuals` for our models. Therefore we have to fetch those names from elements inside *uml:Model* node corresponding to the IFML elements. These can be identified by using *xmi:id* attributes on the elements which are equal to the *base_%* attributes in *IFML:* namespace.

- Similarly, we expect every element to have an id, therefore we will use the already mentioned *xmi:id* values. These will conveniently provide a mapping between IFML model elements and RDF/OWL elements.

- The remaining attributes that belong to IFML elements, and are expected to be transformed to `data properties` can be found as attributes on the elements in *IFML:* namespace.

- The nesting of the elements in the *uml:Model* node indicates that there is *some kind* of relationship between the parent-child elements. However, we do not have the luxury of knowing what kind of association that is, we just have the participants. Therefore we will have to refer to our metamodel in the ontology and infer the relationship from the participants.

```xml
<?xml version="1.0" encoding="windows-1252"?>
<xmi:XMI xmlns:uml="http://www.omg.org/spec/UML/20110701"
        xmlns:xmi="http://www.omg.org/spec/XMI/20110701"
                xmlns:IFML="http://www.sparxsystems.com/profiles/IFML/1.0">
        <xmi:Documentation exporter="Enterprise Architect" exporterVersion="6.5"/>
        <uml:Model xmi:type="uml:Model" name="EA_Model">
                <packagedElement xmi:type="uml:Package"
                        xmi:id="EAPK_E921F46F_CB90_4462_8E67_45E529A41836"
                        name="Poc1">
                        <packagedElement xmi:type="uml:Component"
                                xmi:id="EAID_316F284E_4634_4658_B348_3CD84861F74C"
                                name="Homepage">
                                <nestedClassifier xmi:type="uml:Component"
                                        xmi:id="EAID_E51583A7_A6CE_4916_97F4_424117FD32CE"
                                        name="MainMenu"/>
                                <nestedClassifier xmi:type="uml:Class"
                                        xmi:id="EAID_E69DE1F0_7808_41ea_BECE_C2C8B1BCAB40"
                                        name="Shopping_Cart"/>
                                <nestedClassifier xmi:type="uml:Class"
                                        xmi:id="EAID_B1DB7AC9_8643_4bfa_B145_125D427A0FA3"
                                        name="Recommended_Products"/>
                                <nestedClassifier xmi:type="uml:Class"
                                        xmi:id="EAID_253B61D2_018D_43b1_BF19_F02057525C42"
                                        name="Categories">
                                        <ownedAttribute xmi:type="uml:Port"
                                                xmi:id="EAID_EBFDA47A_2D69_487f_9826_3BB0A70420BA"
                                                name="OnSelectEvent" aggregation="composite"/>
                                </nestedClassifier>
                        </packagedElement>
                </packagedElement>
                <profileApplication xmi:type="uml:ProfileApplication"
                        xmi:id="profileap_43202EE1-F">
                        <appliedProfile xmi:type="uml:Profile"
                                href="http://www.sparxsystems.com/profiles/IFML/1.0#43202EE1-F"/>
                </profileApplication>
        </uml:Model>
        <IFML:ViewContainer base_Component="EAID_316F284E_4634_4658_B348_3CD84861F74C"
            isLandMark="false"
            isXor="false"
            isDefault="true"/>
        <IFML:Menu base_Component="EAID_E51583A7_A6CE_4916_97F4_424117FD32CE"/>
        <IFML:Form base_Class="EAID_E69DE1F0_7808_41ea_BECE_C2C8B1BCAB40"/>
        <IFML:List base_Class="EAID_B1DB7AC9_8643_4bfa_B145_125D427A0FA3"/>
        <IFML:List base_Class="EAID_253B61D2_018D_43b1_BF19_F02057525C42"/>
        <IFML:OnSelectEvent base_Port="EAID_EBFDA47A_2D69_487f_9826_3BB0A70420BA"/>
</xmi:XMI>
```

Listing 1: XMI Representation of PoC Example #1

Another type of relationship is present when modeling `Flows`. In this case, nesting is obviously not present because this is not a parent-child relationship. However, IFML flows have been modeled as *uml:Dependecies*, therefore source and target of the relationship is provided in *supplier* and *client* attributes. An example of such relationship representation can be found in the following snippet on listing 2.

```
    <packagedElement
        xmi:type="uml:Dependency"
        xmi:id="EAID_BB5259EA_0AA0_4962_8BBD_AEB09687FADA"
        name="To_Payement_Flow"
        supplier="EAID_F916316E_5B0E_48ee_8F0B_088213A75B0B"
        client="EAID_AA646F95_9FA8_4d71_A632_4637E22FE5C2"/>
```

Listing 2: Interaction Flow Association Representation

However, as we already discussed in section 6.2.1.3, for some associations, most notably *parameter bindings*, there is no nesting and no *uml:Dependency* to indicate the existence of the relationship. Therefore, we have resulted in using **tagged values** to model this relationship. Since these tagged values are created in a separate namespace, the result looks as we can see in the XMI snippet below on listing 3.

```
<IFML:ParameterBinding
    base_Class="EAID_EAB30D5B_2E2F_4019_8730_7ECBAC8FDC49"/>
    <thecustomprofile:hasTargetInteractionFlowElement
        base_Class="EAID_EAB30D5B_2E2F_4019_8730_7ECBAC8FDC49"
        hasTargetInteractionFlowElement="Price"/>
    <thecustomprofile:hasSourceInteractionFlowElement
        base_Class="EAID_EAB30D5B_2E2F_4019_8730_7ECBAC8FDC49"
        hasSourceInteractionFlowElement="Total"/>
```

Listing 3: Interaction Flow Association Representation

#### 6.3.1.2 RDF/OWL File

The resulting file is expected to be in RDF/OWL format. Format settings are provided by ProtegeAPI that we are using to handle ontologies in our application and since RDF/OWL is the most common format used there is no issue. The goal is to create files that only contain `Individuals` with IFML meta model ontology only imported, as we have been creating them in our proof-of-concept examples (e.g. listing 4).

```
<?xml version="1.0"?>
<rdf:RDF xmlns="https://github.com/PatrikJantosovic/ifml-ontology/poc#"
     xml:base="https://github.com/PatrikJantosovic/ifml-ontology/poc"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:ifml="https://github.com/PatrikJantosovic/ifml-ontology/ifml#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:Ontology rdf:about="https://github.com/PatrikJantosovic/ifml-ontology/poc">
        <owl:imports rdf:resource="https://github.com/PatrikJantosovic/ifml-ontology/ifml"/>
    </owl:Ontology>
    <!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // Individuals
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->
    <!-- https://github.com/PatrikJantosovic/ifml-ontology/poc#Categories -->
    <owl:NamedIndividual
        rdf:about="https://github.com/PatrikJantosovic/ifml-ontology/poc#Categories">
        <rdf:type rdf:resource="https://github.com/PatrikJantosovic/ifml-ontology/ifml#List"/>
        <ifml:name
            rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Categories</ifml:name>
            <ifml:id rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                EAID_BB5259EA_0AA0_4962_8BBD_AEB09687FADA
           </ifml:id>
            <ifml:hasViewContainer
                rdf:resource="https://github.com/PatrikJantosovic/ifml-ontology/poc#Homepage"/>
        <ifml:hasOnSelectEvent
            rdf:resource="https://github.com/PatrikJantosovic/ifml-ontology/poc#CategorySelect"/>
    </owl:NamedIndividual>
</rdf:RDF>
```

Listing 4: RDF File Format - PoC Example #1

### 6.3.2 Implementation Details

In this subsection, implementation details of the transformation application and technical decisions are briefly discussed.

#### 6.3.2.1 Structure

The application is divided into multiple packages. **Api** package contains classes describing IFML elements, a class representing OWL data property structure, and a class representing OWL object property structure. Besides that, it also contains the factory class that instantiates the objects. The **cmd** package provides an entry point to the application through the *picocli* command-line interface and also implements spring configuration loading. Most of the logic resides in **core** package, with an XMI parser class and OWL modifier class, which are classes implementing the logic of reading information from IFML models in XMI format and writing them to the resulting files in RDF/OWL format.

#### 6.3.2.2 Basic Idea

The very basic idea of transformation is explained in the following snippet of the pseudo-code:

```
create new target ontology
import ifml:metamodel ontology to target ontology
for each element in ifml: namespace of xmi file:
    read id from base_% attribute
    read name from child element of uml:model node using id
    read attributes from the element
    create individual
for each individual:
    verify individual type against ifml:metamodel ontology
    add individual axiom to the target ontology
    for each attribute in individual.attributes:
        verify data-property ifml:metamodel ontology
        add data-property axiom to the target ontology
for each individual:
    get nesting relations from xmi file
    get uml:dependency relations from xmi file
    get tagged-values relation from xmi file
    save to individual
for each individual:
    for each relation:
        find object-property in ifml:metamodel ontology
        add object-property axiom to the target ontology
save target ontology to file
```

#### 6.3.2.3 Inferring Relationships

As already mentioned, sometimes we only have participants and their types of relationships. To know which association this is, we use **HermiT** reasoner that is part of ProtegeAPI. At first, we try to fetch `object-property` definition by using classes of these `Individuals`. If such object property exists, we use it. Otherwise, we fetch for all *superclasses* using the reasoner and try to find the appropriate object property in the whole hierarchy.

Furthermore, to simplify the issue with parent-child relationships, children are related to the parent on the parent level. This means that the resulting subject-predicate-object triple is always in the form of parent-relationship-child.

### 6.3.3 Usage

To use the transformation application you will need Java 11 installed and IFML meta model ontology downloaded from the GitHub project: https://github.com/PatrikJantosovic/ifml-ontology to your local computer.

You can fetch the built *ifml2rdf.jar* from the GitHub repository: https://github.com/PatrikJantosovic/ifml2rdf release section. The application can then be run with the following parameters:

- **–path** being the path to the source IFML file
- **–target** being the path where we want to store our RDF/OWL result
- **–iri** being resulting ontology IRI

Furthermore, we should set the following properties in *the application.properties* file, as these are not necessary as input parameters.

- **metamodel.path** being path to IFML metamodel ontology file
- **metamodel.iri** being metamodel IRI, which by default is: https://github.com/PatrikJantosovic/ifml-ontology/ifml

The application uses *Apache log4j2* logging service, to log information to standard output on different levels. Therefore, we can also set the following properties:

- **logging.level.root** being the default application logging level, to avoid useless logs from Spring, we should keep this set to WARN
- **logging.level.com.jantosovic.ifml** being application-specific logs logging level, this should be set to INFO/DEBUG

Sample application.properties file is published alongside the jar file for better usability.

### 6.3.4 Proof Of Concept

Now, we will try to demonstrate the application in our prepared examples.

#### 6.3.4.1 Example #1

For our first example we will execute the application with following arguments:

```
--path="\\ifml2rdf\PoC\poc1-xmi.xml"
--target="\\ifml2rdf\transformation\\poc.owl"
--iri="https://github.com/PatrikJantosovic/ifml-ontology/poc1"
```

And we indeed end up with a **poc.owl** file that contains our imported IFML ontology and our **Individuals** corresponding to the IFML elements as we can see in the following snippets:

```
<!-- https://github.com/PatrikJantosovic/ifml-ontology/poc1#Homepage -->
<owl:NamedIndividual rdf:about="https://github.com/PatrikJantosovic/ifml-ontology/poc1#Homepage">
    <rdf:type
        rdf:resource="https://github.com/PatrikJantosovic/ifml-ontology/ifml#ViewContainer"/>
    <ifml:isDefault
        rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">true</ifml:isDefault>
    <ifml:isLandMark
        rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">false</ifml:isLandMark>
    <ifml:isXor
        rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">false</ifml:isXor>
    <ifml:hasViewElement
        rdf:resource="https://github.com/PatrikJantosovic/ifml-ontology/poc1#Categories"/>
    <ifml:hasViewContainer
        rdf:resource="https://github.com/PatrikJantosovic/ifml-ontology/poc1#MainMenu"/>
    <ifml:hasViewElement
        rdf:resource="https://github.com/PatrikJantosovic/ifml-ontology/poc1#Recommended_Products"/>
    <ifml:hasViewElement
        rdf:resource="https://github.com/PatrikJantosovic/ifml-ontology/poc1#Shopping_Cart"/>
</owl:NamedIndividual>
```

Listing 5: Transformed RDF file - Example #1: Individual structure

On the listing 5 we can see that the structure of the exported file corresponds to the one we have designed in our ontology and later used during the modeling of Example #1 in RDF. We have our named individual of *Homepage* element, described using rdf:type as `ViewContainer` class. Then, we can see our IFML attributes as data properties.

Furthermore, we can see the hierarchy structure of containers represented through `hasViewElement` and `hasViewContainer` relationships. In the following snippet on listing 6, we can see the representation of the association between *Categories* and its related event *OnSelectEvent*.

```
<!-- https://github.com/PatrikJantosovic/ifml-ontology/poc1#Categories -->

<owl:NamedIndividual
    rdf:about="https://github.com/PatrikJantosovic/ifml-ontology/poc1#Categories">
    <rdf:type
        rdf:resource="https://github.com/PatrikJantosovic/ifml-ontology/ifml#List"/>
    <ifml:hasOnSelectEvent
        rdf:resource="https://github.com/PatrikJantosovic/ifml-ontology/poc1#CategorySelect"/>
</owl:NamedIndividual>
```

Listing 6: Transformed RDF file - Example #1: Association representation

And as we can see, when we compare our RDF graph (id and name attributes are omitted for readability) on fig. 6.12 to the one we have created in the RDF proof of concept section on fig. 6.9 we can clearly see that these differ only in selected relationships that were used to relate the parent and child, as the transformation tool always pick parent-hasChild-child relationship. A full RDF diagram of the transformation result can be found in external attachments under the identifier **T1**.

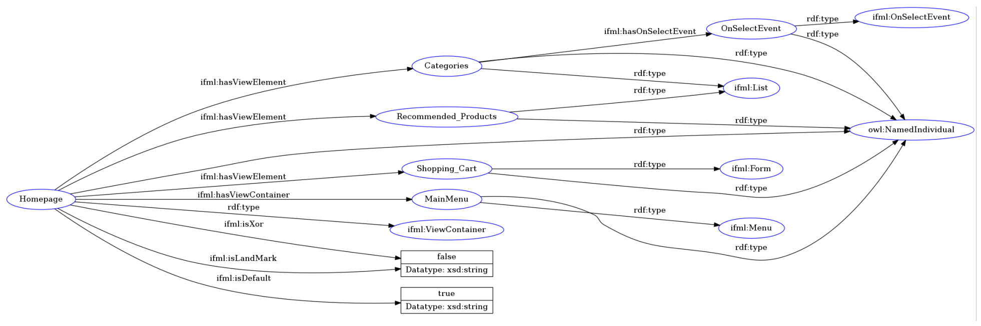


Figure 6.12: Transformed RDF - Example #1

#### 6.3.4.2 Example #2

We have demonstrated that we can create the appropriate `Individuals` with their properties, and basic association that is represented as a nested relationship in the XMI file in our first example. Here, we will try to demonstrate that the application is able to transform the remaining two relationship types which we have described in Implementation Details (section 6.3.2).

Therefore, we run our application with the following parameters:

```
--path="\\ifml2rdf\PoC\poc2-xmi.xml"
--target="\\ifml2rdf\transformation\poc2.owl"
--iri="https://github.com/PatrikJantosovic/ifml-ontology/poc2"
```

And as a matter of fact, we are indeed able to transform even the **flow** and **tagged-value** relationship as we can see on listing 7 and listing 8. Full RDF diagram of the transformation result can be found in external attachments under the identifier **T2**.

```xml
<!-- https://github.com/PatrikJantosovic/ifml-ontology/poc2#To_Payment_Flow -->

<owl:NamedIndividual
    rdf:about="https://github.com/PatrikJantosovic/ifml-ontology/poc2#To_Payment_Flow">
    <rdf:type
        rdf:resource="https://github.com/PatrikJantosovic/ifml-ontology/ifml#NavigationFlow"/>
    <ifml:hasParameterBindingGroup
        rdf:resource="https://github.com/PatrikJantosovic/ifml-ontology/poc2#PriceParameters"/>
    <ifml:hasSourceInteractionFlowElement
        rdf:resource="https://github.com/PatrikJantosovic/ifml-ontology/poc2#Checkout"/>
    <ifml:hasTargetInteractionFlowElement
        rdf:resource="https://github.com/PatrikJantosovic/ifml-ontology/poc2#Pay_For_Products"/>
</owl:NamedIndividual>
```

Listing 7: Transformed RDF - Flow association representation

```
<!-- https://github.com/PatrikJantosovic/ifml-ontology/poc2#Total-Price -->

<owl:NamedIndividual
    rdf:about="https://github.com/PatrikJantosovic/ifml-ontology/poc2#Total-Price">
    <rdf:type
        rdf:resource="https://github.com/PatrikJantosovic/ifml-ontology/ifml#ParameterBinding"/>
    <ifml:hasSourceParameter
        rdf:resource="https://github.com/PatrikJantosovic/ifml-ontology/poc2#Total"/>
    <ifml:hasTargetParameter
        rdf:resource="https://github.com/PatrikJantosovic/ifml-ontology/poc2#Price"/>
</owl:NamedIndividual>
```

Listing 8: Transformed RDF - Tagged-value association representation

### 6.3.5 Evaluation

As we have demonstrated, we are able to transform the IFML models to the RDF/OWL format using our transformation application. This application supports creating all commonly used IFML elements, their attributes, and their most common associations. The application is easy to use, allowing the user to select their target IRI, source file, and target destination of the RDF/OWL file. The application offers multiple levels of logs for debugging and error handling and is well documented both by using Javadoc and higher-level documentation in the readme file.

While there definitely is a lot to improve, this result clearly demonstrates that the automated transformation from IFML to RDF/OWL that adheres to our IFML meta model ontology is possible and working in the described scope.

The application can be found on GitHub:
`https://github.com/PatrikJantosovic/ifml2rdf`.
The reflected version is tagged with **0.0.1-RELEASE** tag in case there is more work done.

Chapter 7

# NSGO4CM Transformation

In this chapter, the transformation from generated RDF/OWL model to the RDF model adhering to the Normalized Systems Gateway Ontology For Conceptual Models is designed.

NSGO4CM is currently in active development and is yet to be published, however, the first prototypes of NS meta model ontology and transformation tool have been provided by the author, Marek Suchánek.

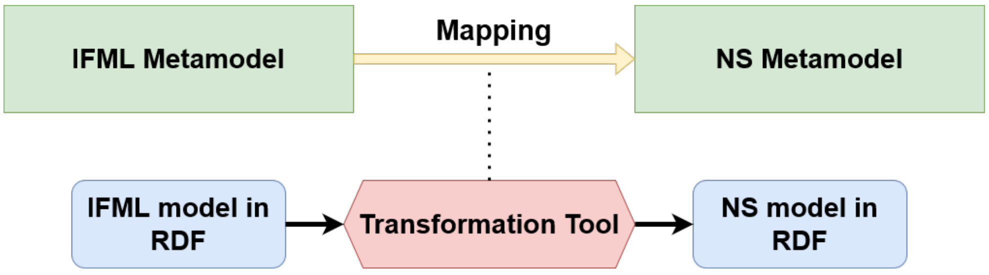


Figure 7.1: IFML to NS Transformation Diagram

According to the unpublished work [29], transformation is done by providing a mapping between IFML metamodel and NS metamodel, as depicted on fig. 7.1. The transformation tool provided to us is a python application with a command-line interface. This tool takes an IFML model (or any supported conceptual model) in ontology format and *Turtle* syntax as input. Furthermore, configuration files in the form of input-output mapping are supplied to the application. Then, SPARQL CONSTRUCT queries are built based on the supplied mapping configuration file and used to transform the input file in our IFML ontology to the NS ontology.

After some experimenting with the provided tool, we have identified the additional steps needed for successful transformation from IFML models into NS elements:

- The NSGO4CM tool expects the input ontology to be in Turtle format, therefore the output from our application should be changed from RDF/XML to Turtle syntax.
- Change transformation application to use *prefixes* instead of full IRIs of IFML elements, resulting in a more readable format, especially when combined with Turtle syntax.
- Prepare sample mapping configuration for a subset of IFML elements and demonstrate the successful transformation.

## 7.1 Transformation Application Changes

As for the change of the syntax, protege API provides a convenient way of changing the output format when saving ontology to file. Therefore, the additional parameter, **model.syntax**, has been added, which can be set in the application.properties. The default value is *TURTLE*, but to keep backward compatibility, another supported option is *RDF/XML*. Other formats have not been supported for now as they are not needed.

Additionally, Protege API also provides a possibility to import, set and use prefixes alongside imported ontologies. Therefore, an **ifml:** prefix is now set for imported meta model ontology, resulting in a cleaner output file. The result of the transformation of our proof-of-concept #1 that we have been using throughout this thesis can be seen in the snippet on listing 9.

As stated before, the application can be found on GitHub:
`https://github.com/PatrikJantosovic/ifml2rdf`.
The version, which supports Turtle syntax and IFML prefix, is tagged as
**0.0.2-RELEASE**.

```
@prefix : <https://github.com/PatrikJantosovic/ifml-ontology/poc1#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ifml: <https://github.com/PatrikJantosovic/ifml-ontology/ifml#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@base <https://github.com/PatrikJantosovic/ifml-ontology/poc1> .

<https://github.com/PatrikJantosovic/ifml-ontology/poc1>
    rdf:type owl:Ontology ;
    owl:imports <https://github.com/PatrikJantosovic/ifml-ontology/ifml> .

:Categories rdf:type owl:NamedIndividual ,
                     ifml:List ;
        ifml:name "Categories"^^xsd:string ;
        ifml:hasOnSelectEvent :OnSelectEvent .

:Homepage rdf:type owl:NamedIndividual ,
                   ifml:ViewContainer ;
        ifml:name "Homepage"^^xsd:string ;
        ifml:isXor "false"^^xsd:string ;
        ifml:isLandMark "false"^^xsd:string ;
        ifml:isDefault "true"^^xsd:string ;
        ifml:hasViewElement :Categories ;
        ifml:hasViewContainer :MainMenu ;
        ifml:hasViewElement :Recommended_Products ,
                            :Shopping_Cart .

:MainMenu rdf:type owl:NamedIndividual ,
        ifml:Menu ;
        ifml:name "MainMenu"^^xsd:string .

:OnSelectEvent rdf:type owl:NamedIndividual ,
        ifml:OnSelectEvent ;
        ifml:name "OnSelectEvent"^^xsd:string .

:Recommended_Products rdf:type owl:NamedIndividual ,
        ifml:List ;
        ifml:name "Recommended_Products"^^xsd:string .

:Shopping_Cart rdf:type owl:NamedIndividual ,
        ifml:Form ;
        ifml:name "Shopping_Cart"^^xsd:string .
```

Listing 9: Transformed Proof-Of-Concept #1 in Turtle syntax

## 7.2 Mapping

As the next step, a sample set of rules for mapping is proposed and some of them are implemented for demonstration purposes. For now, the mapping definition needed for proof-of-concept #1 is proposed. These rules are expected to be revisited, refactored, and expanded in further research.

- IFML elements denoting *the structure* of the application are transformed into *DataElement*. This is mainly *ViewElement* and its descendants such as *Form*, *List*, *Detail*, *Window*.
- Type of the container should be explicitly defined in the *DataOption* belonging to the *DataElement*.
- IFML elements describing *events* and *actions* are transformed into *TaskElement*. This might include types such as *OnSelectEvent, OnSubmitEvent, OnLoadEvent, and Action* and it is specified in the associated *TaskOption*.
- IFML elements representing *data* presented to the user in form of fields are transformed into *Field* entities. This includes all descendants of *ViewComponentPart*. The type is specified in the associated *FieldOption*.
- Name of the IFML elements is transformed into the name attribute of the constructed NS element.
- All the remaining attributes of the IFML elements are transformed into DataOptions or TaskOptions.
- Parent-child associations of containers and view elements can be simulated by using *LinkField*.

On listing 10, a sample mapping file representing a simple transformation from IFML:Form to NS:DataElement is presented. Here, we can see the definition of the input and the expected output. In the first section, some meta-data about the transformation is defined, as well as variables are defined. In the input section, it is set that we expect the *ifml:Form*, with *ifml:name* attribute and we also specify a filter for the Sparql query to include only named elements. In the output section, we have defined that we expect the element to be transformed into *ns:DataElement*, with *ns:DataElement-name* being the name of the element and *ns:DataElement-type* being a primary data element.

To demonstrate the result, an example of *ifml:Form* definition as seen on listing 11 is transformed using the mapping. The result of the successful transformation can be seen on listing 12.

```
{
  :mapping a sbmo:Mapping .
  :mapping rdfs:label "IFML Form element mapping for Data Element" .
  :mapping sbmo:hasGraphPattern :input .
  :mapping sbmo:hasConstructTemplate :output .
  :cls a sbmo:Variable .
  :cls sbmo:preferredName "cls" .
  :cls-name a sbmo:Variable .
  :cls-name sbmo:preferredName "clsName" .
}

:input {
  :cls a ifml:Form .
  :cls ifml:name :cls-name .
  :input sbmo:hasFilter "STRLEN(?clsName) > 0" .
}

:output {
  :cls a ns:DataElement .
  :cls ns:DataElement-name :cls-name .
  :cls ns:DataElement-type "Primary" .
}
```

Listing 10: Mapping of IFML:Form to NS:DataElement

```
###  https://github.com/PatrikJantosovic/ifml-ontology/poc1#Shopping_Cart

:Shopping_Cart rdf:type owl:NamedIndividual ,
                        ifml:Form ;
        ifml:id "EAID_E69DE1F0_7808_41ea_BECE_C2C8B1BCAB40"^^xsd:string ;
        ifml:name "Shopping_Cart"^^xsd:string .
```

Listing 11: IFML:Form representation in Proof-of-Concept #1

```
<https://github.com/PatrikJantosovic/ifml-ontology/poc1#Shopping_Cart>
                                                a ns1:DataElement ;
    ns1:DataElement-name "Shopping_Cart"^^xsd:string ;
    ns1:DataElement-type "Primary" .
```

Listing 12: NS:DataElement representation of IFML:Form from PoC #1

## 7.3 Proof Of Concept

In this section, mapping files for the elements that are present in the first proof-of-concept example are prepared. This includes mapping for `Form`, `ViewContainer`, `List`, `Menu` and `OnSelectEvent` elements.

Unfortunately, there is a problem present with part of the transformation. For the proposed mapping, the ability to generate or calculate a unique uniform resource identifier (URI) should be implemented. For example, imagine having to create a *ns:DataOption* for each *ns:DataElement* to describe a type of UI element. For now, this is not possible, as there is nothing to query for in the input file. That is because it does create a piece of additional information based on the type of the transformed element.

A similar issue naturally occurs when trying to transform a data-property or object-property predicate to the normalized systems element as a unique subject of the RDF triple. The feedback regarding this was provided to the author and it will be resolved in the future versions of the transformation tool.

Ignoring this issue, it is still possible to transform IFML elements into NS elements. The result of transformation from our model (listing 9) can be found on listing 13. Clearly, we can see the result of our IFML elements mapping, and their successful transformation, but, associations between the elements and attributes of the elements are missing in the transformed file.

```
<https://github.com/PatrikJantosovic/ifml-ontology/poc1#MainMenu>
    a ns1:DataElement ;
    ns1:DataElement-name "MainMenu"^^xsd:string ;
    ns1:DataElement-type "Primary" .

<https://github.com/PatrikJantosovic/ifml-ontology/poc1#OnSelectEvent>
    a ns1:TaskElement ;
    ns1:TaskElement-name "OnSelectEvent"^^xsd:string ;
    ns1:TaskElement-type "Primary" .

<https://github.com/PatrikJantosovic/ifml-ontology/poc1#Recommended_Products>
    a ns1:DataElement ;
    ns1:DataElement-name "Recommended_Products"^^xsd:string ;
    ns1:DataElement-type "Primary" .

<https://github.com/PatrikJantosovic/ifml-ontology/poc1#Shopping_Cart>
    a ns1:DataElement ;
    ns1:DataElement-name "Shopping_Cart"^^xsd:string ;
    ns1:DataElement-type "Primary" .

<https://github.com/PatrikJantosovic/ifml-ontology/poc1#Categories>
    a ns1:DataElement ;
    ns1:DataElement-name "Categories"^^xsd:string ;
    ns1:DataElement-type "Primary" .

<https://github.com/PatrikJantosovic/ifml-ontology/poc1#Homepage>
    a ns1:DataElement ;
    ns1:DataElement-name "Homepage"^^xsd:string ;
    ns1:DataElement-type "Primary" .
```

Listing 13: Mapping of Proof-of-Concept #1

However, as we only have unique associations in our proof-of-concept model, we can simulate the URI generation in our transformation mapping files by hard-coding the identifiers. This is obviously not optimal, but this way we can at least demonstrate the result of the future transformation more accurately. The hard-coded identifiers are prefixed with **URI-** to clearly distinguish what should be fixed in future versions of the transformation tool. On listing 15, the result of the transformation can be seen, now containing the parent-child associations of different types of `ViewContainers` through the *LinkField* entity of Normalized Systems. Additionally, attributes of IFML elements, such as `ifml:isXor`, are transformed into *DataOption* entities. The example of a mapping file for the *hasViewContainer* association can be seen on listing 14. All of the mapping files can be found on the attached CD.

```
@prefix rdfs: <https://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <https://www.w3.org/2002/07/owl#> .
@prefix ns: <https://normalizedsystems.org/owl/elements#> .
@prefix sbmo: <https://example.com/sbmo#> .
@prefix ifml: <https://github.com/PatrikJantosovic/ifml-ontology/ifml#> .
@prefix : <https://example.com/ns-mapping/ifml/m-hasViewContainer#> .

{
	:mapping a sbmo:Mapping .
	:mapping rdfs:label "IFML hasViewContainer association mapping for LinkField" .
	:mapping sbmo:hasGraphPattern :input .
	:mapping sbmo:hasConstructTemplate :output .
	:clsA a sbmo:Variable .
	:clsA sbmo:preferredName "clsA" .
	:clsB a sbmo:Variable .
	:clsB sbmo:preferredName "clsB" .
}

:input {
	:clsA ifml:hasViewContainer :clsB .
	:input sbmo:hasFilter "EXISTS { ?clsA ifml:hasViewContainer ?clsB }" .
}

:output {
	"URI-viewContainer" a ns:Field, ns:LinkField .
	"URI-viewContainer" ns:Field-name "hasViewContainer" .
	"URI-viewContainer" ns:Field-dataElement :clsA .
	"URI-viewContainer" ns:Field-targetDataElement :clsB .
}
```

Listing 14: Association hasViewContainer mapping into LinkField

```
@prefix ns1: <https://normalizedsystems.org/owl/elements#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

"URI-onSelectEvent" a ns1:Field,
        ns1:LinkField ;
    ns1:Field-dataElement <https://github.com/PatrikJantosovic/ifml-ontology/poc1#Categories> ;
    ns1:Field-name "hasOnSelectEvent" ;
    ns1:Field-targetDataElement <https://github.com/PatrikJantosovic/ifml-ontology/poc1#OnSelectEvent> .

"URI-viewContainer" a ns1:Field,
        ns1:LinkField ;
    ns1:Field-dataElement <https://github.com/PatrikJantosovic/ifml-ontology/poc1#Homepage> ;
    ns1:Field-name "hasViewContainer" ;
    ns1:Field-targetDataElement <https://github.com/PatrikJantosovic/ifml-ontology/poc1#MainMenu> .

"URI-viewElement" a ns1:Field,
        ns1:LinkField ;
    ns1:Field-dataElement <https://github.com/PatrikJantosovic/ifml-ontology/poc1#Homepage> ;
    ns1:Field-name "hasViewElement" ;
    ns1:Field-targetDataElement <https://github.com/PatrikJantosovic/ifml-ontology/poc1#Categories>,
        <https://github.com/PatrikJantosovic/ifml-ontology/poc1#Recommended_Products>,
        <https://github.com/PatrikJantosovic/ifml-ontology/poc1#Shopping_Cart> .

<https://github.com/PatrikJantosovic/ifml-ontology/poc1#MainMenu> a ns1:DataElement ;
    ns1:DataElement-name "MainMenu"^^xsd:string ;
    ns1:DataElement-type "Primary" .

<https://github.com/PatrikJantosovic/ifml-ontology/poc1#OnSelectEvent> a ns1:TaskElement ;
    ns1:TaskElement-name "OnSelectEvent"^^xsd:string ;
    ns1:TaskElement-type "Primary" .

<https://github.com/PatrikJantosovic/ifml-ontology/poc1#Recommended_Products> a ns1:DataElement ;
    ns1:DataElement-name "Recommended_Products"^^xsd:string ;
    ns1:DataElement-type "Primary" .

<https://github.com/PatrikJantosovic/ifml-ontology/poc1#Shopping_Cart> a ns1:DataElement ;
    ns1:DataElement-name "Shopping_Cart"^^xsd:string ;
    ns1:DataElement-type "Primary" .

"URI-ifml:isDefault" a ns1:DataOption ;
    ns1:DataOption-dataElement <https://github.com/PatrikJantosovic/ifml-ontology/poc1#Homepage> ;
    ns1:DataOption-name "ifml:isDefault" ;
    ns1:DataOption-value "true"^^xsd:string .

"URI-ifml:isLandMark" a ns1:DataOption ;
    ns1:DataOption-dataElement <https://github.com/PatrikJantosovic/ifml-ontology/poc1#Homepage> ;
    ns1:DataOption-name "ifml:isLandMark" ;
    ns1:DataOption-value "false"^^xsd:string .

"URI-ifml:isXor" a ns1:DataOption ;
    ns1:DataOption-dataElement <https://github.com/PatrikJantosovic/ifml-ontology/poc1#Homepage> ;
    ns1:DataOption-name "ifml:isXor" ;
    ns1:DataOption-value "false"^^xsd:string .

<https://github.com/PatrikJantosovic/ifml-ontology/poc1#Categories> a ns1:DataElement ;
    ns1:DataElement-name "Categories"^^xsd:string ;
    ns1:DataElement-type "Primary" .

<https://github.com/PatrikJantosovic/ifml-ontology/poc1#Homepage> a ns1:DataElement ;
    ns1:DataElement-dataOption "URI-ifml:isDefault",
        "URI-ifml:isLandMark",
        "URI-ifml:isXor" ;
    ns1:DataElement-name "Homepage"^^xsd:string ;
    ns1:DataElement-type "Primary" .
```

Listing 15: Simulated Mapping of Proof-of-Concept #1

Chapter 8

# Case Study

In the final chapter, results are demonstrated in a case study using a simple generic e-shop as our modeled domain. Normally, IFML comes as a complement to the content domain model, typically in form of a UML diagram. However, here the focus is solely on the part that is represented in IFML such as the structure of the user interface and interaction flow describing processes on the e-shop.

## 8.1 Requirements

The goal is not to model the whole e-shop but to demonstrate the capabilities of our solution on a subset of the most common modeling scenarios of IFML. Therefore, the following requirements have been selected to be modeled as our case study:

- **F1** - Homepage of the e-shop, consisting of recommended products section, list of categories, main navigation menu of the e-shop, search field, and a simple snippet of a shopping cart. This is almost identical to already established proof-of-concept #1.

- **F2** - The checkout process, starts from the shopping cart, with payment and order confirmation. For this, a similar model to the model of proof-of-concept #2 is used.

- **F3** - The process of adding an item to the shopping cart with quantity selection.

- **F4** - The product detail page with multiple fields describing the product.

- **F5** - The process of searching for a product.

## 8.2 IFML Representation

These requirements are represented in the following IFML model on fig. 8.1. This model can also be found in external attachments with **C1** identifier and in the attached Enterprise Architect project.

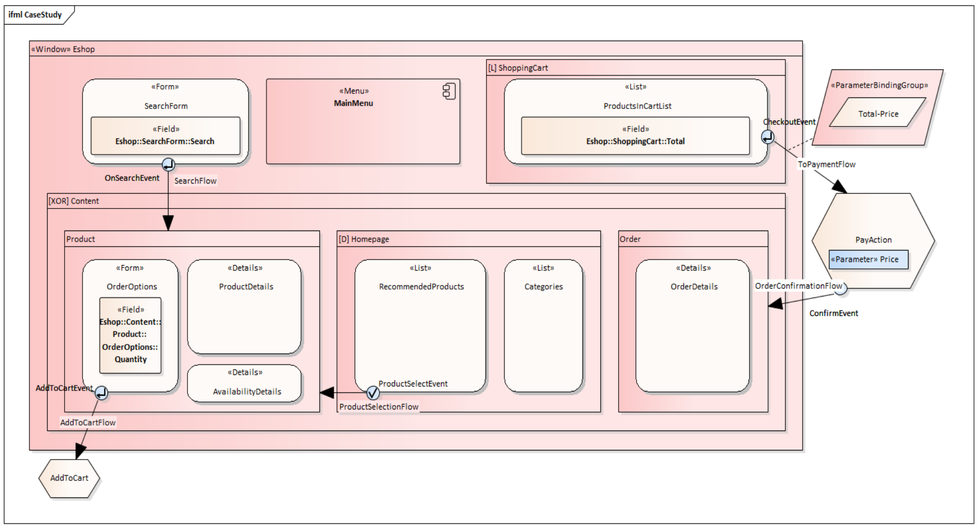


Figure 8.1: E-shop case study - IFML representation

The web application is modeled as a single window, with a sub-container for a shopping cart, a menu element for a navigation menu, and a form that consists of a search field. Furthermore, a content parent container is created with the Xor attribute, meaning that only one child of this element is rendered at the same time, virtually creating the shared template of the website for our specific content pages. This structure effectively covers our **F1** requirement.

One of our content pages is a simple product page with details on the product, availability of the product, and the form allowing us to add the product to the cart in the selected quantity. Interaction with this form is modeled using an event and activity resulting in the item being added to the shopping cart. This covers the requirements **F3** and **F4**.

The checkout process (requirement **F2**), is implemented almost identically to the proof-of-concept #2. The process starts with on submit event using the checkout button in the shopping cart, passing the total price of products as a parameter to the payment action. On completion of this action, the user is redirected to the order summary detail page.

Lastly, the process of searching for a product is triggered by the on-submit event from the search form after the user provides the textual input to the field. On a successful search, the user is directly redirected to the product page following the navigation flow. This covers our last **F5** requirement.

## 8.3 RDF Representation

As the next step, the IFML model is exported in XMI format and transformed into RDF/OWL representation using our application implemented in Chapter 6. The whole result can be found as an RDF graph in external attachments identified as **C2**, however, ontology is already pretty complex and large which makes it harder to read for humans.

In the following paragraphs, the result is dissected and it is demonstrated that structures are all present and that they represent the requirements we have set.

For requirement **F1** and **F4**, hierarchy of containers denoting the structure is described through parent-child relationship in form of *hasViewContainer* and *hasViewElement* object properties. Visualized on fig. 8.2.

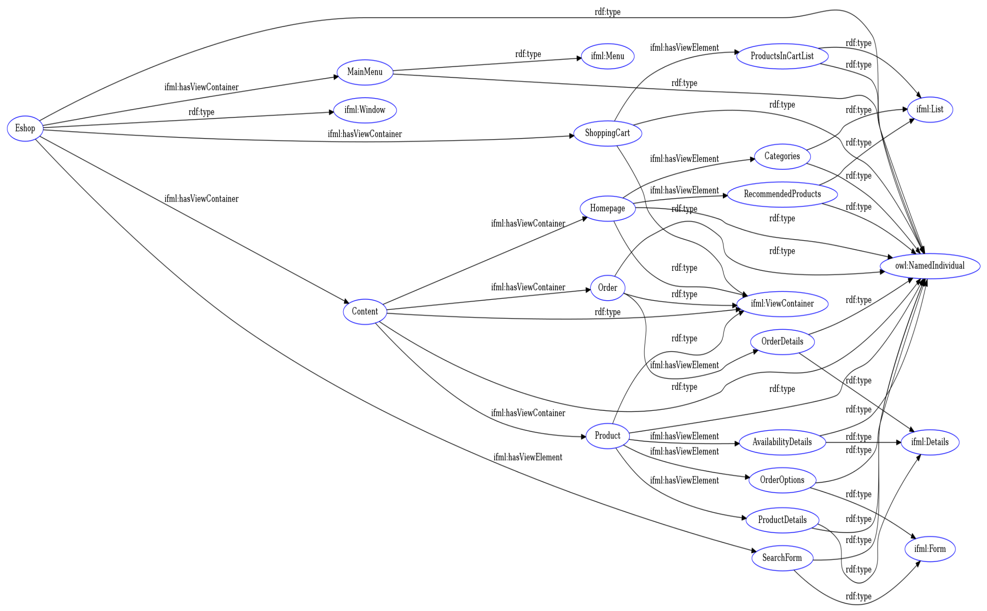


Figure 8.2: Transformed Case-Study - F1 & F4 Requirements

Requirement **F2** is very similar to already mentioned proof-of-concept #2 model. Two navigation flows are present, describing the process from checkout through payment activity to the order confirmation. The visualization can be found on fig. 8.3.

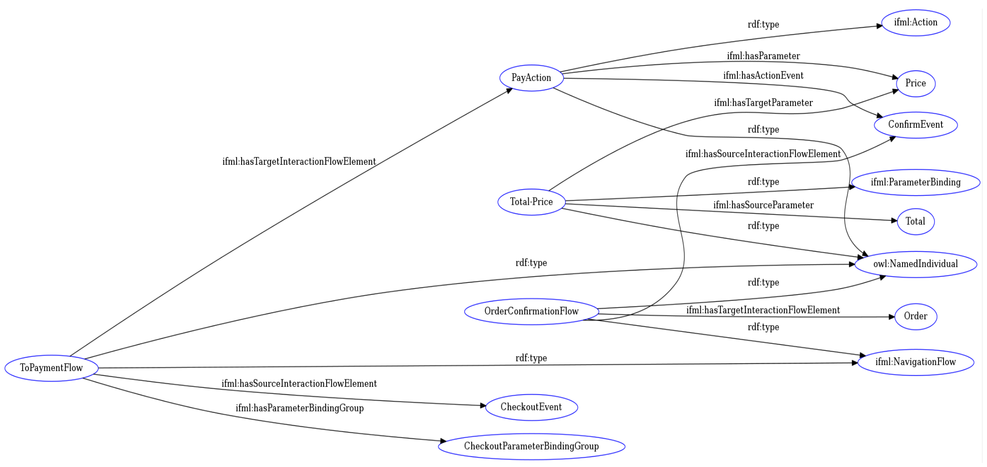


Figure 8.3: Transformed Case-Study - F2 Requirement

For requirement **F3**, the subset of ontology that can be found on fig. 8.4 is relevant. Clearly, the core of the process, which is the navigation flow starting from *AddToCartEvent* to *AddToCartAction* is present.

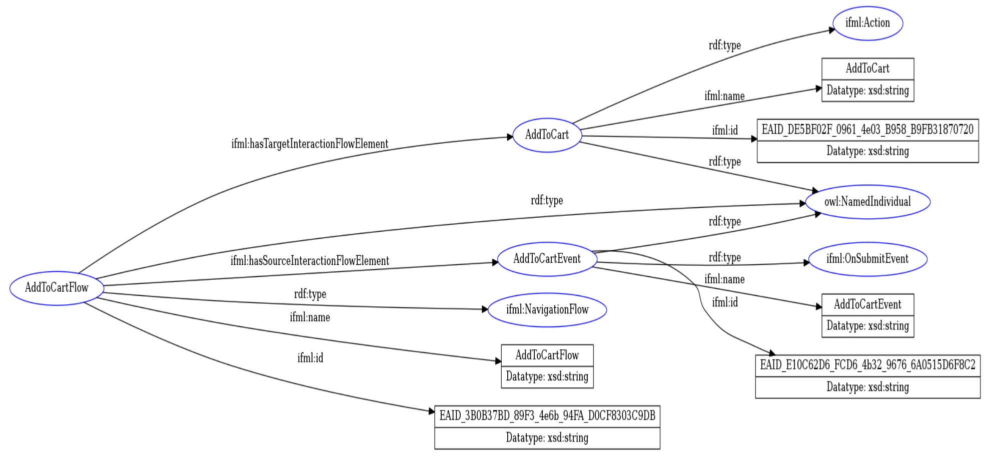


Figure 8.4: Transformed Case-Study - F3 Requirement

And finally, the process of searching for a product (requirement **F5**) is also present in the transformed ontology as seen on the snippet on fig. 8.5. The *SearchForm* has a view element event, which triggers a navigation flow called *SearchFlow* resulting in the navigation to the product page.

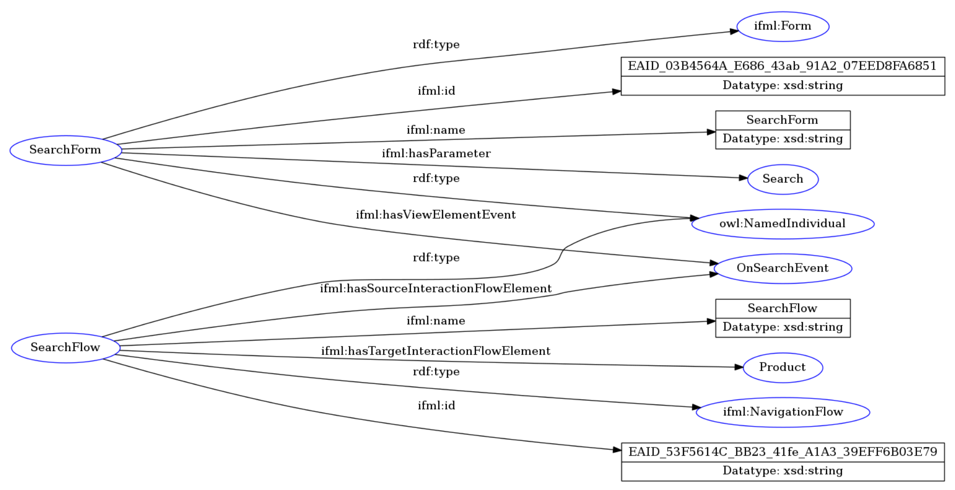


Figure 8.5: Transformed Case-Study - F5 Requirement

## 8.4 Final Result

In the case study, it was demonstrated that the automatic transformation application is successfully applied even on larger models resulting in complex ontologies, supporting different kinds of associations and all IFML elements that are present in IFML meta model ontology are being created as named individuals.

This representation can now be transformed into the format that is defined by Normalized Systems Gateway Ontology for Conceptual Models and therefore transformed into Normalized Systems elements. But, since NSGO4CM development is still in progress, the final transformation of this case study will not be performed as not all constructs and elements of IFML are mapped yet. This topic will require further research once NSGO4CM and the transformation tool are in the stable version.

Chapter 9

# Evaluation

In this chapter, goals from Section 1.2 are revisited and evaluated. It is also important to note, that the results were obtained in multiple iterations as the problem domains were quite unexplored and some of the objectives were readjusted during the work.

In Chapter 2, Chapter 3, Chapter 4 and Chapter 5, the theoretical foundations of the problem domains were explored and documented. We have acquainted ourselves with Normalized Systems Theory, Interaction Flow Modeling Language, and Semantic Web technologies. Furthermore, we have explored tools and APIs for these domains. Therefore, we can conclude that we have achieved our first objective (**O1**).

The second objective (**O2**) was to design an ontology of Interaction Flow Modelling Language metamodel and implement it. This was successfully prepared in Section 6.1, using *Protege* ontology editor, and is one of the outputs and contributions of our thesis. As a next step, in objective number three (**O3**), we wanted to design a process of IFML modeling in the form of ontologies using our prepared metamodel. The implementation of the metamodel and its application in modeling was successfully demonstrated on two proof-of-concept examples in Section 6.2.

In the fourth objective (**O4**), the goal was to automatize transformation from IFML models in form of diagrams into the IFML model ontologies. This was achieved by implementing a simple transformation application in Chapter 6. Afterward, in Section 6.3.4 we have successfully demonstrated that the transformation is indeed functional in our proof-of-concept examples.

Moving towards Normalized Systems, the fifth objective (**O5**) was to design transformation between IFML models and NS models. Furthermore, the transformation tool should be implemented according to the proposed mapping of the metamodels. However, during the work on this thesis, it became clear that the tool is already part of the NSGO4CM solution. Therefore, only mapping was proposed and implemented in the form of mapping files that are provided as input to the existing transformation solution. The mapping was developed only for a part of the IFML metamodel and would require further research, therefore we can conclude that the objective number five (**O5**) was achieved only partially.

The last objective, (**O6**), was to illustrate the complete solution in a complex case study. Therefore, a simple e-shop was modeled in IFML and then successfully transformed using our implemented solution. The final transformation to the NS elements was not performed as the NSGO4CM mapping is not prepared in a sufficient range. We can consider objective number 6 partly achieved.

Chapter 10

# Conclusion

In this thesis, we explored the possibilities of transforming an Interaction Flow Modeling Language model into Normalized Systems elements by using Normalized Systems Gateway Ontology for Conceptual Models.

After acquainting ourselves with problem domains, we started by implementing the IFML meta model ontology. As a next step, a set of rules for modeling using the prepared metamodel was proposed. This allowed us to implement these rules in transformation application and therefore automatize the whole process of IFML model transformation into an ontological IFML model.

Afterward, the goal of transforming these IFML models into Normalized Systems elements was pursued. And even though NSGO4CM is still in development, we were able to show promising results when transforming our ontological models. After getting acquainted with NSOGO4CM, sample mapping between NS and IFML meta-models was proposed and implemented. After that, we successfully demonstrated the transformation using provided tool.

There have been multiple issues encountered along the way, mostly because of the exploratory nature of this thesis. There have been slight adjustments made to the IFML metamodel when transforming it into ontology, in order to simplify its usage and avoid some inconsistencies. Enterprise Architect IFML implementation has proven to be imperfect as there was no standard way of representing some of the associations in a machine-readable format, which we however solved by providing additional information to our models. This could possibly be avoided by using different modeling tools and it could therefore lead to changes in our transformation application, however, we managed to work around the issue somewhat successfully. Furthermore, our own application had to be prepared in two iterations, resulting in two versions. This is because once we acquired the NSGO4CM transformation tool, we had to modify our own application output to make it compatible with the expected input format.

As for further research, numerous possibilities appeared from the incompleteness of our research as well as from the issues we have encountered along the way. Another IFML modeling tool can be explored, and in case of a better output format, the transformation could be simplified. Another possibility is to finish the IFML-NS metamodel mapping and demonstrate the transformation on more complex models. Furthermore, additional research in Normalized Systems can be pursued by looking into UI-specific Normalized Systems expanders and structures and combining them with the IFML models.

To summarize the contributions and achievements of this thesis, the most notable one is the exploration itself and lessons learned along the way, and the demonstration that this transformation is indeed possible. Hopefully, this provides a positive outlook for future research into NS, NSGO4CM, and its possibilities related to UI modeling. Another significant output is the IFML meta model ontology which is publicly available and can be used in further research. Additionally, we prepared the transformation application of IFML diagram models exported from Enterprise Architect in XMI format into ontologies.

# Bibliography

[1] Oorts, G.; Huysmans, P.; et al. Building Evolvable Software Using Normalized Systems Theory: A Case Study. In *2014 47th Hawaii International Conference on System Sciences*, 2014, pp. 4760–4769, doi: 10.1109/HICSS.2014.585.

[2] Mannaert, H.; Verelst, J.; et al. The transformation of requirements into software primitives: Studying evolvability based on systems theoretic stability. *Science of Computer Programming*, volume 76, no. 12, 2011: pp. 1210–1222, ISSN 0167-6423, doi:https://doi.org/10.1016/j.scico.2010.11.009, special Issue on Software Evolution, Adaptability and Variability. Available from: `https://www.sciencedirect.com/science/article/pii/S016764231000208X`

[3] Mannaert, H.; Verelst, J.; et al. Towards evolvable software architectures based on systems theoretic stability. *Software: Practice and Experience*, volume 42, no. 1, 2012: pp. 89–116, doi:https://doi.org/10.1002/spe.1051, `https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.1051`. Available from: `https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.1051`

[4] Mannaert, H.; De Cock, K.; et al. On the realization of meta-circular code generation: the case of the normalized systems expanders. In *Proceedings of the Fourteenth International Conference on Software Engineering Advances (ICSEA)*, volume 2019, 2019, pp. 171–176.

[5] Brambilla, M.; Fraternali, P. Interaction Flow Modeling Language Specification. 2015. Available from: `https://www.omg.org/spec/IFML/1.0/PDF`

[6] Koch, N. UML-based Web Engineering. 2016. Available from: `http://uwe.pst.ifi.lmu.de/index.html`

[7] Kroiß, C.; Koch, N.; et al. *UWE Metamodel and Profile.* 2011. Available from: `http://uwe.pst.ifi.lmu.de/download/UWE-Metamodel-Reference-v1.9.pdf`

[8] Ceri, S.; Fraternali, P.; et al. Web Modeling Language (WebML): a modeling language for designing Web sites. *Comput. Networks*, volume 33, 2000: pp. 137–157.

[9] Hamdani, M.; Butt, W. H.; et al. A Systematic Literature Review on Interaction Flow Modeling Language (IFML). ICMSS 2018, New York, NY, USA: Association for Computing Machinery, 2018, ISBN 9781450354318, p. 134–138, doi:10.1145/3180374.3181333. Available from: `https://doi.org/10.1145/3180374.3181333`

[10] Ed-Douibi, H.; Bruneliere, H. Modeling software application front-ends: introducing the open source IFML graphical editor.... In *EclipseCon North America 2015 - Modeling Symposium*, San Francisco, United States, Mar. 2015. Available from: `https://hal.inria.fr/hal-01146785`

[11] Laaz, N.; Wakil, K.; et al. Comparative Analysis of Interaction Flow Modeling Language Tools. *Journal of Computer Science*, 2018.

[12] Enterprise Architect User Guide Web Page. 2021. Available from: `https://sparxsystems.com/enterprise_architect_user_guide/15.2/model_domains/ifml_tech.html`

[13] Bernaschina, C.; Comai, S.; et al. IFMLEdit.org: Model Driven Rapid Prototyping of Mobile Apps. In *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 2017, doi:10.1109/MOBILESoft.2017.15.

[14] Acerbis, R.; Bongio, A.; et al. Model-Driven Development Based on OMG's IFML with WebRatio Web and Mobile Platform. In *Engineering the Web in the Big Data Era*, edited by P. Cimiano; F. Frasincar; G.-J. Houben; D. Schwabe, Cham: Springer International Publishing, 2015, ISBN 978-3-319-19890-3, pp. 605–608.

[15] Hitzler, P.; Krötzsch, M.; et al. OWL 2 web ontology language primer. *W3C recommendation*, volume 27, 2009.

[16] Singh, G.; Bhatia, S.; et al. OWL2Bench: A Benchmark for OWL 2 Reasoners. In *The Semantic Web – ISWC 2020*, edited by J. Z. Pan; V. Tamma; C. d'Amato; K. Janowicz; B. Fu; A. Polleres; O. Seneviratne; L. Kagal, Cham: Springer International Publishing, 2020, ISBN 978-3-030-62466-8.

[17] OWL2 Axioms Web Page. 2021. Available from: `https://www.w3.org/TR/owl2-syntax/#Axioms`

[18] OWL2 Entities Web Page. 2021. Available from: `https://www.w3.org/TR/owl2-syntax/#Entities.2C_Literals.2C_and_Anonymous_Individuals`

[19] OWL2 Guide Web Page. 2021. Available from: `https://www.w3.org/TR/2004/REC-owl-guide-20040210`

[20] Horridge, M.; Jupp, S.; et al. A practical guide to building owl ontologies using protégé 4 and co-ode tools edition1. 2. *The university of Manchester*, volume 107, 2009.

[21] Resource Description Framework (RDF) Model and Syntax Specification. 1999. Available from: `https://www.w3.org/TR/PR-rdf-syntax/`

[22] Resource Description Framework Serialization Syntax. 2011. Available from: `https://www.w3.org/wiki/RdfSyntax`

[23] RDF/XML Syntax Specification. 2004. Available from: `https://www.w3.org/TR/REC-rdf-syntax/`

[24] Pérez, J.; Arenas, M.; et al. Semantics and Complexity of SPARQL. *ACM Trans. Database Syst.*, volume 34, no. 3, sep 2009, ISSN 0362-5915, doi:10.1145/1567274.1567278. Available from: `https://doi.org/10.1145/1567274.1567278`

[25] Musen, M. A. The protégé project: a look back and a look forward. *AI matters*, volume 1, no. 4, 2015: pp. 4–12.

[26] Protege Web Page. 2021. Available from: `https://protege.stanford.edu/`

[27] ProtegeAPI web page. 2022. Available from: `https://protegewiki.stanford.edu/wiki/ProtegeOWL_API_Programmers_Guide`

[28] IFML Examples web page. 2022. Available from: `https://www.ifml.org/ifml-examples/`

[29] Suchánek, M. Towards a Normalized Systems Gateway Ontology for Conceptual Models, [Dissertation Thesis (Unpublished)].

APPENDIX A

# External attachments

External attachments are part of this work for better readability of modeled diagrams. They are labeled according to the following list. All attachments are contained in an *attachments* directory on the external device that has been submitted with this thesis.

- R1 - Proof of Concept Example #1 in RDF Format
- R2 - Proof of Concept Example #2 in RDF Format
- I1 - Proof of Concept Example #1 as IFML diagram
- I2 - Proof of Concept Example #2 as IFML diagram
- T1 - Transformation Proof of Concept Example #1 in RDF Format
- T2 - Transformation Proof of Concept Example #2 in RDF Format
- C1 - Case Study IFML diagram
- C2 - Case Study - RDF graph

Appendix B

# Acronyms

**API** Application Programming Interface

**EA** Enterprise Architect

**IRI** Internationalized Resource Identifier

**UI** User Interface

**IFML** Interaction Flow Modeling Language

**NS** Normalized Systems

**NST** Normalized System Theory

**HTML** Hyper Text Markup Language

**MDG** Model Driven Generation

**W3** World Wide Web

**OWL** Web Ontology Language

**RDF** Resource Description Framework

**BPMN** Business Process Model and Notation

**UWE** UML-based Web Engineering

**WebML** Web Modelling Language

**XML** Extensible Markup Language

**XMI** XML Metadata Interchange

**NSGO4CM** Normalized Systems Gateway Ontology for Conceptual Models

**URI** Uniform Resource Identifier

Appendix **C**

# Contents of enclosed CD

| | |
|---|---|
| `readme.txt` | the file with CD contents description |
| `attachements` | the directory with external attachments |
| `src` | the directory of source codes |
|     `metamodel` | the directory with containing IFML ontology |
|     `application` | the directory with source files of application |
|     `mapping` | the directory with IFML-NS mapping files |
|     `ifml.eap` | enterprise architect project with examples |
|     `thesis` | the directory with LaTeX source code of the thesis |
| `thesis.pdf` | the thesis text in PDF format |