From Rigorous Requirements and User Interfaces Specifications into Software Business Applications: The ASL Approach

Ivo Miguel Torrado Gamito

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Alberto Manuel Rodrigues da Silva

Examination Committee

Chairperson: Prof. Daniel Jorge Viegas Gonçalves

Supervisor: Prof. Alberto Manuel Rodrigues da Silva

Member of the Committee: Dr. João Paulo Pedro Mendes de Sousa Saraiva

January 2021

Abstract

Software applications have been developed with multiple programming languages together with specific software libraries and frameworks and deployed on various software and hardware infrastructures. We introduce and discuss the ASL language (short for “Application Specification Language”) that combines constructs from two previous languages: ITLingo RSL and OMG IFML. ASL specifications are strict and rigorous sentences that allow to define both requirements and user interfaces aspects of software applications in a consistent and integrated way. Alike RSL, and differently from IFML, ASL is a controlled natural language with a textual concrete syntax.

This research proposes an approach, called “ASL approach” that produces source code artifacts for a popular Python Web framework, Django. The approach consists of several tasks including the generation of UI textual specifications (model-to-model transformation) and crucial components for Django (model-to-code transformation). We apply and evaluate these methods through two case studies: MyTinyBlog, a typical blog application and RiverCure Portal, a web platform that improves water resources management and protection.

Keywords: Requirements Engineering, Model-driven engineering, Web engineering, ITLingo RSL, OMG IFML, ITLingo ASL, RiverCure Portal.

Resumo

As aplicações de software têm sido desenvolvidas com várias linguagens de programação juntamente com bibliotecas e frameworks específicas e instaladas em diferentes infraestruturas de software e hardware. Nesta dissertação, apresentamos e discutimos a linguagem ASL (abreviação de "Application Specification Language") que combina duas linguagens anteriores: ITLingo RSL e OMG IFML. As especificações ASL são frases estritas e rigorosas que permitem definir os requisitos e os aspetos das interfaces do utilizador das aplicações de software de forma consistente e integrada. Tal como o RSL, e contrariamente ao IFML, o ASL é uma linguagem natural controlada com uma sintaxe textual concreta.

Esta investigação propõe uma abordagem, a "ASL approach", que produz artefactos de código-fonte para uma popular framework de Python, Django. A abordagem consiste em várias tarefas, incluindo a geração de especificações textuais de interfaces de utilizador (transformação de modelo para modelo) e componentes cruciais para o Django (transformação de modelo para código). Aplicamos e avaliamos esses métodos através de dois casos de estudo: o MyTinyBlog, uma aplicação típica de blog e o RiverCure Portal, uma plataforma web que visa melhorar a gestão e proteção dos recursos hídricos.

Palavras-chave: Engenharia de Requisitos, Model-driven engineering, Web engineering, ITLingo RSL, OMG IFML, ITLingo ASL, RiverCure Portal.

Acknowledgments

First, I would like to express my sincere gratitude to Professor Alberto Silva for his guidance. I would not have made it through my master's degree without his support.

Next, I want to thank the entire team in the RiverCure Project. Everyone contributed with their knowledge to make this project successful and I wish them the best of luck for this and all their other projects. Special thanks to Jorge Marques, whom I developed the RiverCure Portal with.

Many thanks to my parents, siblings and friends who supported me throughout my studies.

I also acknowledge Fundação para a Ciência e Tecnologia (FCT) for supporting the work reported in this dissertation.

Table of Contents

List of figures

List of Tables

List of Acronyms

ANEPC	Autoridade Nacional de Emergência e Proteção Civil
API	Application Programming Interface
ASL	Application Specification Language
BPMN	Busines Process Model and Notation
CBV	Class-Based View
CLOC	Count Lines of Code
CNL	Controlled Natural Language
CRM	Customer Relationship Management
CRUD	Create, Read, Update and Delete
DSL	Domain Specific Language
DSRM	Design Science Research Methodology
DTM	Digital Terrain Model
FR	Functional Requirement
GCBV	General Class-Based View
GPL	General Programming Language
IFML	Interaction Flow Modeling Language
MDE	Model-Driven Engineering
MTB	MyTinyBlog
MTV	Model-Template-View
MVC	Model-View-Controller
M2C	Model-to-code
M2M	Model-to-model
NFR	Non-Functional Requirement
NL	Natural Language
OMG	Object Management Group
RCP	RiverCure Portal

RE Requirements Engineering

RSL Requirements Specification Language

SCM Supply Chain Management

SRS Software Requirements Specification

SVARH Sistema de Vigilância e Alerta de Recursos Hídricos

UC Use-Case

UI User Interface

VPN Virtual Private Network

1. Introduction

In this chapter, we explain the context and the goals of this project. We also present the followed research methodology and the dissertation structure.

1.1. Context

Currently, developers use expressive programming languages, software libraries and frameworks that help them develop a multitude of software applications. However, they have to master multiple details of these tools and technologies, which are complex, require long learning curves, and raise challenges like the need to create appealing and cross-platform user interfaces, and the need to deal with cross-cutting concerns like scalability, performance, security and others [1, 2]. In this scope, the importance of requirements engineering (RE) has been crucial to the development and management of software, and to reduce software errors at the early stages of the development process [3]. RE has had a crucial role in different stages of software engineering and has provided a variety of approaches [3]. RE practices have been essential to give a broad understanding of the problem-domain before starting any sort of effort toward the design, development and deployment of a given solution, as well as to prevent rework costs [4,5]. RE has been crucial for the success of a project and has dealt with socio-technical challenges like the adoption of elicitation techniques, communication difficulties, and the management of conflicting and ambiguous requirements [6].

In this scope, the ITLingo initiative intends to mitigate problems that arise when writing requirements. RSL is a controlled natural language that helps writing requirements and test specifications in a systematic, rigorous, and consistent way [7,9]. RSL includes a rich set of constructs logically arranged in views according to concerns that exist at different abstraction levels, such as stakeholders, actors, data entities, use cases, goals, use case tests [7,8,9,10]. Using a model-driven approach to transform RSL specifications, Gonçalo Pereira showed that it is possible to generate web applications to an Angular/ASP.Net framework [11]. However, that approach also showed some limitations, namely it is inflexible in what concerns the fine tuning and better specification of the software application's UIs. This approach has a predefined set of user interfaces used in the final application. On the other hand, Interaction Flow Modeling Language (IFML) is an OMG (Object Management Group) standard modeling language in the field of software engineering. IFML allows to define platform-independent models of the user interfaces of software applications. IFML describes the structure and the behaviour of the applications as perceived by end-users [12].

In this context, we introduce and discuss the design of the ASL specification language ("Application Specification Language") that combines constructs from these languages (further details in the next section): ITLingo RSL [7,8,9,10] and IFML [12]. Like with ITLingo RSL, the ASL specifications (or ASL models) are strict and rigorous sentences. However, ASL is comprehensive enough to specify user interface (UI) aspects, as inspired by the concepts found in modeling languages like IFML. ASL

gathers characteristics and advantages from both RSL and IFML. Likewise RSL, and differently from IFML (that is a visual modeling language), ASL is a controlled natural language with a textual concrete syntax (that is the reason we named it as a “specification language” instead of a “modeling language”). Finally, beyond the rigorous and systematic specification of software applications, we also show that it is possible to take advantage of these specifications to semi-automatically generate software applications following a model-driven engineering approach: this means that with appropriate tools, an ASL user can create define and then web applications, which are be generated through automatic transformations techniques, from ASL rigorous specifications.

1.2. Goals

In this dissertation, we propose an approach to improve the RE process by mitigating some of its problems, namely in what concerns the specification and validation of requirements. This proposal is a model-driven approach, so, not only documents artifacts are considered, but they are also central in the way the software applications are developed. Similarly, to other approaches in ITLingo, ASL deals with business data concepts, but also improves the specification and the extensibility of such elements. When it comes to the resulting generated application, we can go further and explore front-end specifications providing details that are in the same range of detail of other approaches, such as those based on IFML.

The building of web applications in ASL is achieved through two steps: (i) the generation of UI specifications (model-to-model transformation); (ii) the generation of Django framework (model-to-code transformation), using the ASL specifications that was generated in the first step. This is the core part of this project that is explained in Chapters 4 and 5. Subsequently, we evaluate this approach, using two different applications with different domains, purposes, and complexity: (i) the MyTinyBlog application; and (ii) the RiverCure Portal, a web-based system that allows specialized users to manage information regarding water resources captured by distinct sources, such as hydrometric sensors, and use it to multiple objectives, the main one being defining simulations for climate-change events, like floods [13].

1.3. Research methodology

The research used in this project is based on the Design Science Research Methodology (DSRM), as discussed by several authors. This methodology is usually described in six steps [14]:

- Problem identification and motivation which is about defining the specific research problem and justifying the value of a solution for it. In this investigation, we look at problems that start from ambiguous requirements and lead to undesired results in IT projects which usually deal with said requirements using a variety of frameworks and languages.
- Objectives definition which is about defining objectives for a solution to the problem, taking into account what is possible and feasible. In this dissertation, we propose an approach to

improve the RE process by mitigating some of its problems, namely in what concerns the specification and validation of requirements.

- Design and development, the core part of the projects that use this methodology and usually shows the architecture and explains the solution. In this project, it can be explained in two main steps: (i) the generation of UI specifications (model-to-model transformation); (ii) the generation of Django framework (model-to-code transformation), using the ASL specifications that were generated in the first step.
- Demonstration by applying the solution in at least one or more instances of the problem debated: we do this by applying the ASL based approach to two case studies with different features and complexity levels: MyTinyBlog and RiverCure Portal.
- Evaluation which measures how well the artifact developed supports a solution for problem defined, by using objective metrics and comparing the solution to others in order to take conclusions about the developed work. We are going to evaluate our solution with metrics regarding the generated code in model-to-model model-to-code transformations, their usage in deployed systems and a questionnaire answered by RiverCure Portal users.
- Communication that deals with communicating all the previous parts to audiences such as researchers and professionals in the fields that the project problem and solution may involve. We communicate all the previous parts in this dissertation.

1.4. Dissertation structure

This document is composed by six chapters and is organized in the following way: Introduction, Contextualization, Related work, ASL Language, ASL Approach and Key Transformations, Demonstration and Conclusion. Chapter 1, Introduction, introduces the context, the goals of the project along with a brief discussion of the research methodology and document structure. Chapter 2, Background, introduces the important conceptual and technological concepts of this investigation including: model-driven engineering, requirements engineering, Natural Language, Domain Specific Languages, General Programming Languages, Controlled Natural Languages, RSL, IFML, Xtext, Xtend and Django. Chapter 3, Related Work, presents and discusses some approaches and tools that have improved how the community has produced software applications, either developed by the industry or the academia. Chapter 4, ASL Language, introduces the ASL language architecture and its main features, supported by a simple running example. Chapter 5, ASL Approach and Key Transformations introduces and explains the ASL approach and its model-to-model and model-to-code transformations. Chapter 6, Validation, demonstrates and analyses two case studies, where we applied the ASL approach and a comparison to the related work. In Chapter 7, Conclusion, we present a final analysis of the developed work, the contributions for this project and future work.

2. Background

This chapter introduces important concepts and technologies related to this project, namely: MDE (model-driven engineering), RE (requirements engineering) and related practices. We also discuss classes of languages such as: Natural Language, Domain Specific Languages, General Programming Languages, Controlled Natural Languages, RSLingo and IFML. Finally, we present and discuss important technologies for this project: Xtext, Xtend and Django.

2.1. Model-driven approaches

Model-driven engineering (MDE) involves the adoption of languages and transformation engines to address the diversity and complexity of software platforms and frameworks [15]. In the scope of MDE, we consider a model as an abstraction of a system often used to replace the system under study [16]. MDE aims to raise the abstraction level of software specifications and increase automation in software development. Using executable model transformations, a model can be transformed into another (lower level) model until it can be transformed or generated into (programming language) artifacts, or it can be directly executed by some interpretation engine [16].

System requirements are the description of what services and features the system shall provide, as well as its quality attributes and other constraints [5]. These requirements reflect the needs of different stakeholders, like customers, end-users, but also software engineers. System requirements are often broadly classified as functional (FRs), non-functional requirements (NFRs) and constraints. They are statements of features and services the system shall provide and may define how the system responds to its users' inputs, what outputs to generate. FRs may be defined in multiple NFRs, like use cases or scenarios. On the other hand, non-functional requirements define the cross-cutting quality attributes of the system, such as availability, performance, usability, or security. Finally, constraints are requirements that can affect the product itself or the involved development process, and can be defined as a technology, legal or process constraints.

This distinction still does not make the requirements definition easy because there are requirements difficult to distinct between non-functional and functional; also, some requirements often generate or constraint other requirements.

The software requirements specification (SRS) is an official statement of what the system developers should implement. It should include both the user requirements for a system and a detailed specification of the system requirements. Even if sometimes a requirements document is out of date in a short period of time (for example in agile development), it is still important to formally specify the system requirements [17].

The document of requirements may have a diverse set of users, such as [5]:

- System Customers: Specify the requirements and check if all their needs for the system are written
- Development Managers: Use the requirements to plan a bid for the system and to plan the system development process
- System Engineers: Use it to understand what system is to be developed
- System Test Engineers: After developed, they can use the document to develop validation tests and test the system
- System Maintenance Engineers: Use the document to provide maintenance to the system

Natural language (NL) is often used to write system requirements specifications as well as user requirements. User requirements mostly define “how a user will interact with a system and they expect of it” [18], e.g. what happens when the user selects an action on the screen. However, because system requirements are more detailed than user requirements, natural language specifications can become confusing and hard to understand. NL understanding relies on the readers and writer’s interpretation, leading to misunderstandings. NL is over flexible, making it possible for a user and a developer referring to the exact same requirement in completely different ways. Because of these and other related problems, requirements specifications written in natural language may lead to mistakes when building a software product. These mistakes can become very expensive depending on the timing or the feature it affects [19].

A domain-specific language (DSL) is a computer language specialized to a particular application domain. It provides a notation and is based on the relevant concepts and features of that domain. DSLs are small languages, focused on a particular aspect of a software system. We cannot build a whole program with a DSL, but we can often use multiple DSLs in a system mainly written in a general-purpose language. DSL can be much easier to program with than a traditional programming language. It may also improve the communication with domain experts, becoming an important tool [20,21,22].

There are general programming languages (GPL) that are used for any number of purposes to solve any number of problems. Generally, they can be used to implement anything that is computable with a Turing machine [23]. These languages are used for what they are used due to different reasons. They offer different features and are also optimized for the tasks that are relevant in the respective domains. Using C, programmers can benefit from a language that combines advantages of low-level and higher-level languages (arrays, pointers…), which can be compiled or executed on any operating system. At the same time, they can choose an object-oriented language as Java when creating modular programs and reusable code. The more specific the tasks are, the more useful specialized languages are. A Domain-Specific Language is simply a language that is optimized for a given class of problems, like the SQL is optimized for solving relational algebra problems [23].

There’s a gap between a natural language and a formal language. Formal languages are languages that are designed by people for specific applications. For example, the notation that mathematicians use is a formal language that is particularly good at denoting relationships among numbers and

symbols. Controlled natural languages (CNL) help mediating these languages. CNLs are engineered subsets of natural languages whose grammar and vocabulary have been restricted in a systematic way to reduce both ambiguity and complexity of full natural languages. Some requirements for these languages are simplicity, a standardized format to give coherence and uniformity to all sentences; short and active voice style, and a limited vocabulary [24].

2.1.1. ITLingo RSL

ITLingo RSL (or just RSL for brevity) is a specification language created to mitigate problems that arise when writing requirements. RSL is a controlled natural language that helps writing requirements and test specifications in a systematic, rigorous, and consistent way. RSL includes a rich set of constructs logically arranged in views according to concerns that exist at different abstraction levels, such as stakeholders, actors, data entities, use cases, goals, use case tests [7,8,9,10]. RSL constructs are logically classified according to two cross-cutting dimensions: abstraction levels and RE specific concerns. According to the abstraction level, the constructs can be used to define businesses, applications, software or even hardware systems. According to the RE concerns dimension, the constructs are classified in the following aspects: active structure, behaviour, passive structure, requirements, tests, relations and sets, and others [7,9].

System (isFinal vs isReusable)	Concerns / Abstract Levels	Active Structure (Subjects)	Behavior (Actions)	Passive Structure (Objects)	Require-ments	Tests	Other	Relations & Sets
	Business	Stakeholder	ActiveElement (Task, Event)	DataEntity	Goal QR Constraint FR UseCase UserStory	Acceptance CriteriaTest UseCaseTest DataEntityTest StateMachine Test	GlossaryTerm Risk Vulnerability Stereotype IncludeAll IncludeElement	SystemsRelation Requirements Relation TestsRelation SystemElements Relation ActiveFlow SystemView SystemTheme
	Application	Actor	StateMachine	DataEntityCluster				
	Software			DataEnumeration				
	Hardware							
	Other							

Figure 1 - Classification of RSL constructs: abstraction levels versus RE specific concern [7]

Spec. 1 illustrates a simple example of an RSL specification that defines the actor “Blog Editor”, whom participates in the use-case “Manage Blog Posts”, which involves the management of the data entity “Blog Post”.

```
Actor aU_Editor "Blog Editor": User
DataEntity e_Post "Blog Post": Document [
  attribute Id "Post ID": Integer [isNotNull isUnique]
  attribute State "Post State": DataEnumeration enum_PostState
  attribute Title "Post Title": String (30) [isNotNull]
  attribute Body "Post Body": Text
  [...] ]

UseCase uc_1_ManagePost "Manage Blog Posts": EntitiesManage [
  actorInitiates aU_Editor
  dataEntity e_Post
  actions Create, Read, Update, Delete]
```

Spec 1. Simple example of a RSL specification.

2.1.2. IFML

Interaction Flow Modeling Language (IFML) is a standard modeling language in the field of software engineering. IFML allows us to define platform independent models of graphical user interfaces (GUIs) of software applications. IFML describes the structure and the behaviour of the applications as perceived by end-users [12]. IFML brings benefits to the development process of application front-ends, namely [25]: (i) supports the specification of application front-ends with different perspectives (the connection with the business logic, the data model, and the graphical presentation layer); (ii) isolates the front-end specification from implementation-specific details; (iii) separates the concerns between roles in the interaction design; and enables the communication of UI design to non-technical stakeholders.

IFML was developed by WebRatio and inspired by the previous WebML notation [26], as well as by other experiences in the Web modeling field [12]. IFML intends to solve a problem mentioned in the introduction: the variety of hardware devices and software platforms and, consequently, the complexity of designing and developing software applications.

IFML supports the specification of the following perspectives [12]: UI structure, UI content specification, events, events transition specification and parameter binding. The UI structure specification consists of the UI containers, while the UI content specification focus on the data contained. The events specification consists of the definition of events that may affect the UI while the specification of events transition defines the changes to apply after those events occur. Finally, specifications of parameter binding consist of the definition of the input-output dependencies between view components and between view components and actions. Figure 2 shows a simple example of an IFML model.

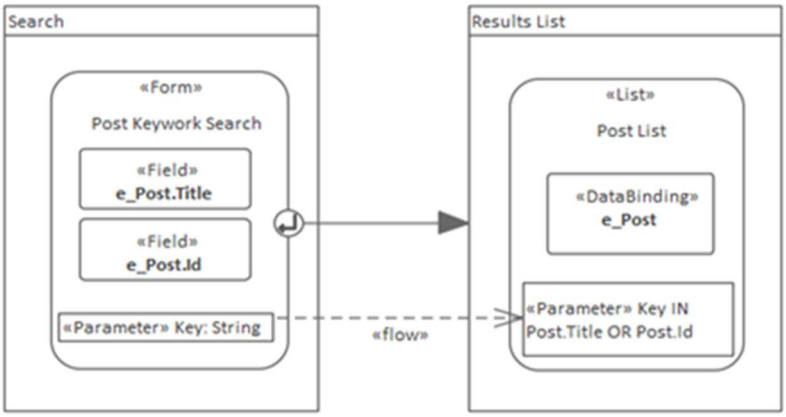

Figure 2 - Search Posts: IFML model

2.2. Technologies

2.2.1. Xtext and Xtend

Xtext is a framework for development of programming and domain-specific languages. With Xtext we can define a language using a powerful grammar language. As a result, we get a full infrastructure, including parser, linker, type checker, compiler, as well as editing support for Eclipse, any editor that supports the Language Server Protocol and a web browser [27]. One of the main features of Xtext is that Xtext can build full-featured text editors for both general-purpose and domain-specific languages. Xtext editors have already been implemented for JavaScript, VHDL, Xtend, and many other languages [28].

Xtend is a statically typed programming language which translates to comprehensible Java source code. Syntactically and semantically Xtend has its roots in the Java programming language. Most of its concepts are very similar to Java, that is, classes, methods, interfaces, etc. Xtend include many language concepts that are especially beneficial when processing models. For example, it offers template strings which are ideal to generate executable code from a given model [29].

Differently from Java, In Xtend, classes may have more than one constructor by using the keyword “new” [30]:

```
class MyClass extends AnotherClass {

   new(String s) {
      super(s)
   }

   new() {
     this("default")
   }
 }
```

Spec 2. Xtend syntax example [30].

2.2.2. Django

Used by some of the largest websites in the world (Instagram, Pinterest, Reddit, Bitbucket, etc.), Django is one of the most used web development frameworks. It is based on Python, the 3rd most popular programming language in 2020, according to the Tiobe Index [31].

As any other framework, it has its own advantages and disadvantages. Starting with the advantages: Being written in Python, lots of useful external libraries and packages are provided and easily imported to a Django project [32]. In Django, more emphasis is placed on explicit programming rather than implicit programming, making it one of the ideal frameworks for applications that require rapid changes. Django provides many useful and advanced functionalities like object-relational mapping (ORM), database migrations, admin panel and user authentication and forms [33].

Django also enables rapid development by having the MTV architecture. The developers can work in parallel and integrate their progress easily. The compatibility with some of the powerful machine

learning libraries like PyTorch or NumPy, makes developers choose Django over other lightweight frameworks, such as Flask, when deploying machine learning models is required [32].

On the other hand, the server processing times can be an issue for Django. For smaller websites that can only run on short bandwidth, this framework may be less viable than others [32]. The functionalities we mentioned for advantages can lead developers to make more errors, sometimes more often than with other popular frameworks, like Flask, which only provides the minimum required for web applications [34].

2.2.3. Django REST Framework

In the current days, talking web development often includes front-end frameworks that were not available when Django was first released. Currently there are several dedicated JavaScript front-end frameworks such as React, Angular, Vue, Ember. Each of them brings several benefits to web applications, but we can still choose to build the front-end around Django template language [35], using template languages (HTML5, CSS), adding Bootstrap [36] and jQuery [37] features. One of the most used front-end frameworks used with Django is React, a JavaScript library for building user interfaces [38]. It provides a clear interface between front-end and back-end logic and makes the user experience smoother, with less loading times and less perceived transitions between pages.

Given the rapid rate of change in front-end libraries, developers often need to create a REST API (Application Programming Interface), which is a software intermediary that allows two applications to talk to each other. APIs can support multiple front-ends written in different languages or frameworks, which comes in great use when we want our Django web application to support different platforms, like with different mobile operating systems [39].

Django REST Framework is the most popular choice to build REST services from Python/Django packages [39] and these are some of its core advantages [40]: a web browsable interface that provides a user-friendly descriptive page for all REST services (e.g., input parameters, options); Integrated authentication mechanisms to restrict access to REST services like OAuth2, HTTP Signature; flexible and sophisticated serializers designed to work with complex data relationships.

2.2.4. Django projects architecture

Django handles most of the HTTP request and response cycle. Developers need only focus on processing the HTTP request, and Django provides tools to make even that easy [41].

Django projects are often structured in the same way, which makes it simpler to work with. Its structure is described in the literature according to the Model-View-Controller (MVC) or Model-View-Template (MVT) architecture. The main difference between the two is how the various concepts are coupled together. Django projects aren’t truly MVC. MVC is strict about which part of the framework can communicate with another. In MVC architecture, the Controller then either tells Model to make changes and update the View or returns a View based on a Model. In this pattern, the view corresponds to the template displayed in the website [41].

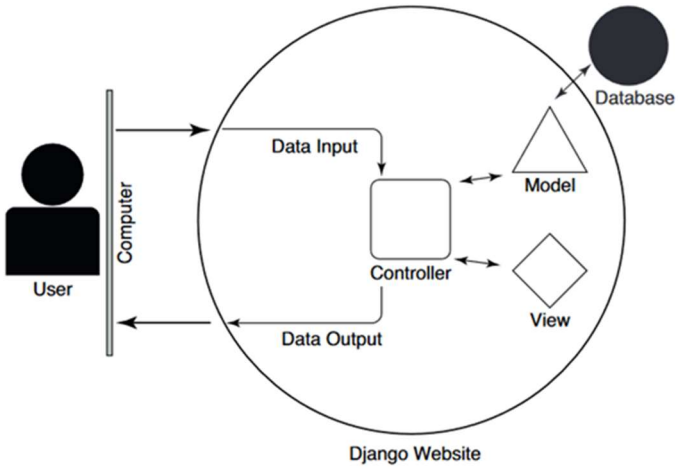

Figure 3 - Django Architecture seen as MVC [41]

MTV is looser and allows for communication between all the parts of the app. In frameworks like Django, developers don't have to write any code related to fetching data from the database and mapping it with the URL (the controller part). This is handled by the framework itself. When a user makes an HTTP request, the corresponding view performs a query on the Model and collects the result set from the Model. The framework will then create a view based on the data and send it to the user. The view fills the result in a template and sends it to the user. [41]

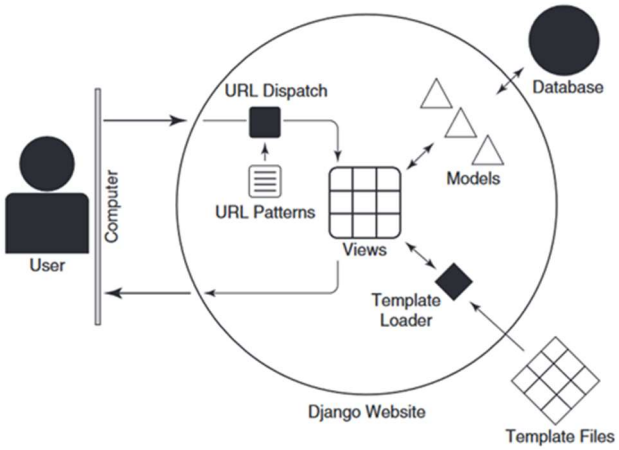

Figure 4 - Django Architecture seen as MVT [41]

2.2.5. Django Admin

Django Admin is a powerful application to manage contents of a relational database linked to a Django project [40,42]. It takes minimal effort to set it up and its model-centric interface is user-friendly. Its use is often limited to administrators of the system; however, it allows a decent customization that is beneficial to applications who are mainly focused in CRUD operations [40,42]. Later, in a case study (Chapter 5) we are going to explore some customizations options for Django Admin Panel.

Using Django Admin, we can deal with Django permissions. All models in a Django project are given a set of permissions to manipulate object model instances [40]. This set of permissions is composed of:

‘add’,’ change’, ‘read’, ‘delete’. These actions are equivalent to the CRUD operations mentioned before: ‘add’ is equivalent to ‘create’ and ‘change’ is equivalent to ‘update’.

These permissions can be linked to the Django user system, based on the “django.contrib.auth” package built in to the Django framework.

Before a user is authenticated in a Django application, he is recognized as an AnonymousUser while navigating the site. Once he authenticates himself, Django recognizes him as a User. There are subtypes of users [40]:

- Superuser: The most powerful user with permissions to create, read, update, and delete data in the Django admin, which includes model records and other users.
- Staff: A user marked as staff can access the Django admin. But permissions to create, read, update, and delete data in the Django admin must be given explicitly to a user. By default, a superuser is marked as staff.
- Active: All users are marked as active if they’re in good standing. Users marked as inactive are not able to authenticate themselves, a common state if there’s a pending post-registration step (e.g., confirm email) or a user is banned, and you don’t want to delete his data.

Django offers the concept of Groups to organize sets of users with the same set of permissions. The permissions we described previously can be assigned to users individually or groups. This is done through Permission model records. Then we can assert these permissions on URL, view methods or even on templates content [40].

```
class BlogPostCreateView(LoginRequiredMixin,UserPassesTestMixin, CreateView):

[...]

    def test_func(self):
        if self.request.user.groups.filter(name='Blogger').exists():
            return True
        else:
            return False
[...]
```

Spec 3. Example of permissions assertion in a Django Generic Class-Based View.

```
  [...]

 <div class="navbar-nav">

       {% if user.is_authenticated %}

       {% if user|has_group:"Administration" or user.is_superuser %}

  [...]
```

Spec 4. Example of permissions assertion in a Django Template.

3. Related Work

This chapter presents some approaches and tools that have improved how the community has produced software applications, either developed by the industry (e.g., Microsoft Power Apps, Mendix, Outsystems, WebRatio, or Quidgest Genio) or by research settings (e.g., EMF on Rails, ADM, XIS or ITLingo RSL).

Table 1 - Related Work tools and their development setting

Tool	Development Setting
Microsoft Power Apps	Industry
OutSystems Studio	
Mendix Studio	
Quidgest Genio	
WebRatio/IFML	Research
EMF On Rails	
ADM	
XIS	
ITLingo Studio (RSL)	

3.1. Industry Related Work

Mendix Studio is a commercial platform designed to enable different groups of people to create software that delivers business value. It was founded in the early 2000s with the belief that software development could be improved with a paradigm shift [43]. Mendix builds a wide range of transactional, event-driven, and adjacent applications for all kinds of industries [43]. According to Mendix perspective, it is becoming harder to keep up to date with the evolving number of programming tools and languages across the spectrum [43]. To reduce the development effort and to improve the feedback loop, Mendix follows a model-driven approach that includes tools like Mendix Studio and Mendix Studio Pro. These tools provide visual drag-and-drop features for UI, data, logic, and navigation using no-code or low-code development [43].

OutSystems Studio is another commercial platform for low-code rapid application development with advanced capabilities for enterprise mobile and web apps [44]. In 2001, OutSystems recognized that a vast majority of software projects were failing due to deadlines not being met or budges being overrun. Therefore, OutSystems Studio was created. It is an integrated development environment that covers the entire development lifecycle, namely: development, quality assurance, deployment, monitoring and management [44].

Quidgest Genio allows users to define architectures, models, configurations of projects through a graphical interface that generates the source code [69]. The generated code follows certain patterns that accommodate technological solutions for a diverse set of business areas, including technology, engineering, banking, and finances [69]. It includes automatic testing and manipulation of code which provides a better performance to this platform users.

3.2. Research Related Work

EMF on Rails proposes an approach that combines MDE with automation frameworks for web development like Spring Roo [45]. It uses ATL, a rule-based declarative model transformation language, where “transformations are specified by mapping object patterns from the source model into patterns of the target model”.

WebML (Web Modeling Language) is a domain-specific language for designing complex, distributed, multi-actor, and adaptive applications deployed on the Web and Service-Oriented Architectures using Web Services [46]. WebML provides graphical, yet formal, specifications, embodied in a complete design process, which can be assisted by visual design tools. It was extended to cover a broader spectrum of frontend interfaces, thus resulting in the Interaction Flow Modeling Language (IFML), adopted as a standard by the OMG. Formerly known as the WebML, it is now IFML because it is no longer limited to web development but also used for mobile apps [46].

ADM (Ariadne Development Method) is another approach with the primary goal of accelerating the development of web systems [68]. Like ASL, it offers constructs to specify these systems making use of Labyrinth++. This tool allows the specification of all the components for web systems and includes a pattern language. Those patterns are organized according to the nature of the problem they solve and make the development of solution easier for less-experienced web developers [68].

XIS is a research project that has developed and evaluated mechanisms and tools to produce business applications more efficiently and productively than it was done [47]. XIS intends to reduce costs and improve the fulfilment of the requirements in software production. XIS approach defends that the most significant effort in a project shall not be in the implementation phase; these activities shall be performed almost automatically, based on high-level and platform-independent specifications. Defining the right specifications shall be the main effort of the developers. XIS also defends a model-driven approach for designing interactive systems at a platform-independent level, considering its modeling languages (i.e., the XIS* languages) that are defined as UML profiles [48,49,50]. The approach discussed in this paper gathers the benefits from the tools and approaches mentioned

above. For example, like the IFML, it supports a platform-independent description of graphical user interfaces.

ITLingo RSL uses a model-driven approach to transform RSL specifications. The approach has a predefined set of user interfaces used in the final application that is based in an Angular/ASP.Net framework [11]. This approach uses the Xtext and it is the closest approach to the one presented in this dissertation, especially because it is part of the ITLingo initiative and due to the fact RSL and ASL share many constructs.

Table 2 - Analysis of the support languages to each tool/approach

Tools	Languages(s)	Meta-language	Concrete Syntax (Visual/Textual/…)	Aspects Supported				
				Data	UI	UseCases	Tests	Workflow
Mendix Studio	Mendix Language	Property format	Visual	Yes	Yes	Yes	Yes	Yes
OutSystems Studio	OutSystems Language	Property format	Visual	Yes	Yes	Yes	Yes	Yes
Quidgest Genio	Genio	Property format	Form-based	Yes	Yes	No	Yes	Yes
WebRatio/IFML	IFML	UML IFML	Visual	Yes	Yes	No	No	Yes
EMF On Rails	ATL	Ecore	Visual	Yes	No	Yes	No	No
ADM	-	Labyrint++	Textual	Yes	No	Yes	No	Yes
XIS	XIS2, XIS-Mobile, XIS-Web, …	UML Profile	Visual	Yes	Yes	Yes	No	No
ITLingo Studio	RSL	Grammar (Xtext)	Textual	Yes	No	Yes	Yes	Yes
	ASL	Grammar (Xtext)	Textual	Yes	Yes	Yes	No	No

Table 3 - Features analysis of each tool/approach

Tools	Languages(s)	Language Support	M2M Transformations		M2C Transformations	
			Technology	Transformations	Transformation Specification	Target Framework
Mendix Studio	Mendix Language	Property format	Mendix UI Framework	*	Property Framework	*
OutSystems Studio	OutSystems Language	Property format	Outsystems UI Framework	*	Property Framework	.Net or Java stacks
Quidgest Genio	Genio	Property format	-	-	Property Framework	MVC architectures (C#, Java, C++, etc.)
WebRatio/IFML	IFML	IFML WebML	-	-	-	Independent
EMF On Rails	ATL	*	*	*	*	Spring Roo
ADM	-	HTML, XML, SMIL, RDF	Ariadne Tool	*	*	Markup Languages
XIS	XIS2, XIS-Mobile, XIS-Web	Sparks Enterprise Architect	XIS Framework	Domain BusinessEntities Architectural UseCases NavigationSpace InteractionSpace	*	WebSQL+ HTML5, JavaScript,CSS Android
ITLingo Studio	RSL	Xtext	Xtend	UseCases Data Entities	Xtend	ASP.NET/Angular
	ASL	Xtext	Xtend	UseCases Data Entities User Interfaces	Xtend	Django

4. ASL Language

This chapter introduces the ASL language and its main features. The discussion is supported by a simple running example named “MyTinyBlog” application, as follows:

MyTinyBlog

MyTinyBlog is a simple web application that allows a blog editor to setup and manage his own blog. The blog administrator manages MyTinyBlog users (includes groups and permissions) and categories. The blog editor may add posts to the blog. Each blog post has a title, a body, the creation date, and the author. A post can be classified by a given category and can be in one of the following states: "Draft" or "Public". Only public posts are visible to the blog's audience (i.e. the readers). Readers can read and submit comments of a public post but can only edit or delete their own comments. They can also add a “like” and share posts.

Figures 5 and 6 illustrate some models of the MyTinyBlog application: the domain model (in Figure 5) and the use-cases model (in Figure 6). The main features of this application involve managing blog posts through typical create, read, update, and delete (CRUD) operations.

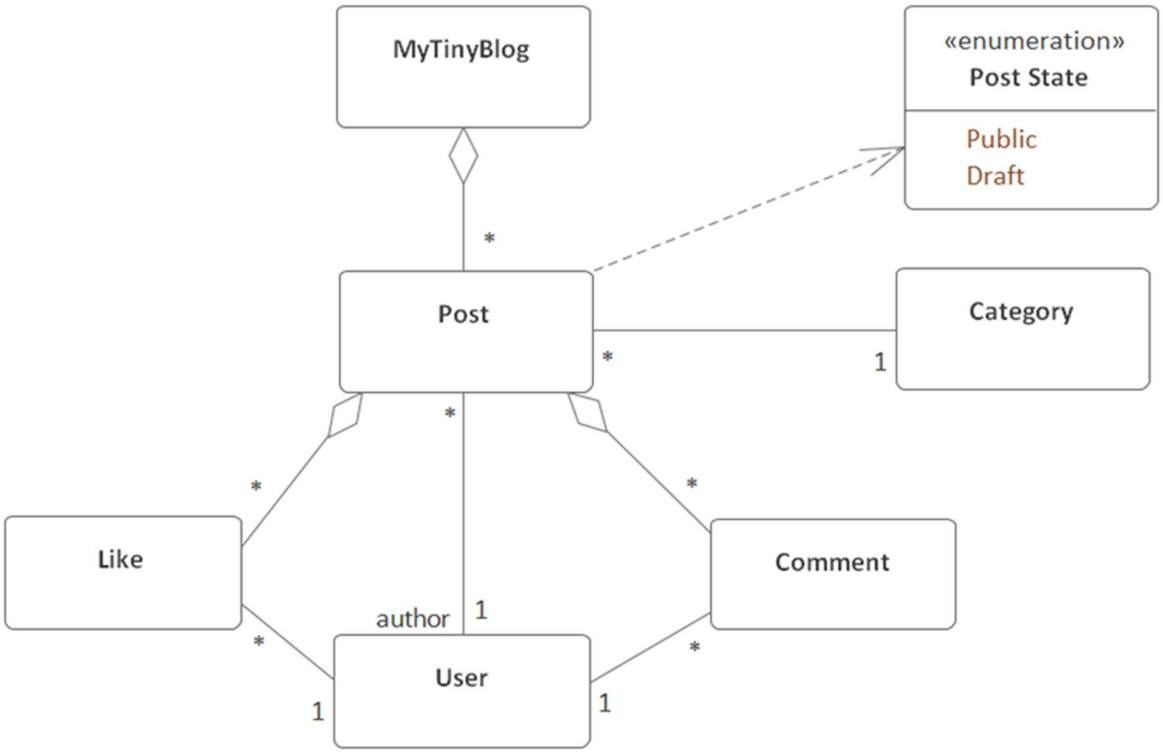

Figure 5 - MyTinyBlog data model (UML class diagram)

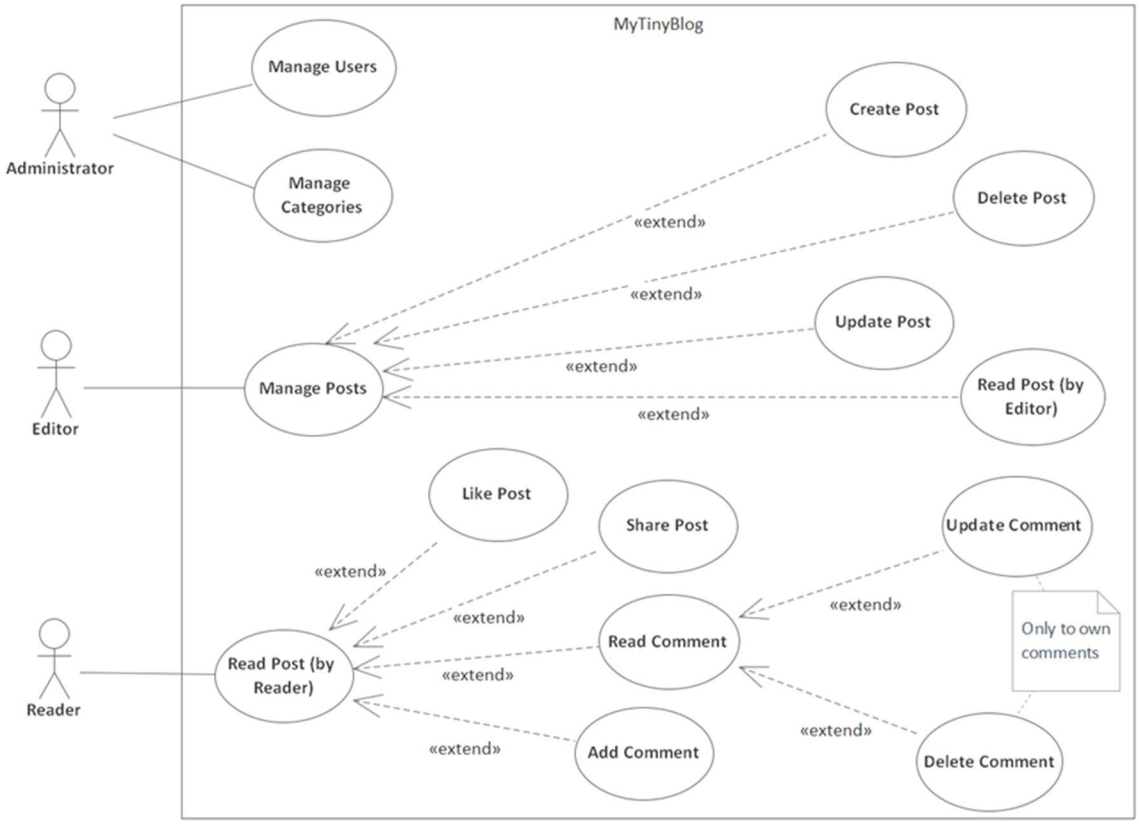

Figure 6 - MyTinyBlog use cases model (UML use cases diagram)

4.1. ASL Architecture

The ASL language combines the main aspects of the RSL and the IFML languages to support the specification of software applications systematically and rigorously. These applications can be considered “business applications, in which data is a core asset, and support several business activities like planning, forecasting, control, coordination, decision making and operational activities [51]. Popular classes of software business applications are e-commerce, ERP (enterprise resource planning), CRM (customer relationship management), SCM (supply chain management).

Table 4 - Classification of ASL constructs: abstraction levels versus RE specific concern

Concerns / Abstract Levels	Active Structure	Behaviour (Actions)	Passive Structure (Objects)	Requirements	User Interfaces
Business		ActiveElement (Task, Event)	DataEntity		
Application	ContextActor		DataEntityCluster		UIAction
Software			DataEnumeration	UseCase	UIViewContainer UIViewComponent UIViewComponentPart
Hardware					
Other					

4.2. Data Entities

ASL adopts and extends the definition of the DataEntity construct as defined initially in RSL [8,9]. DataEntity is the construct used to define domain concepts or information entities such as goods, people, or business transactions. A DataEntity denotes an individual structural entity that might include the specification of attributes, foreign keys and other data constraints [8]. A DataEntity can be classified by type and subtype. The types are the following [70]: (1) **Parameter**, which can include data that is specific to an industry or business; (2) **Reference**, simple reference data, which is required to operate a business process; (3) **Master**, data assets of the business, usually reflects more complex data (e.g., customers, vendors, projects); (4) **Document**, worksheet data that might be converted into transactions later (e.g., invoices); and (5) Transaction, the operational transaction data of the business (e.g., paid invoices). The subtypes are [70]: **Regular** and **Weak**.

ASL Data Entities and Data Entities Clusters can be enriched with the element “Tag” that provide more information to the generation algorithm. A Tag has a name and a value. The Tags presented in the next table change the output of the ASL model-to-code transformations. The Inline affects the layout of interfaces where two data entities have a relationship. Tags with name and value “User” map a data entity and respective attributes with the user profile in the final application.

Table 5 - ASL important Tags Names and Values

Name	**Value**
Inline	Stacked
	Tabular
User	User

In the MyTinyBlog example, we define the following data entities, as also suggested in Figure 5: Blog, Category, User, Blog Post, Comment and Like (see Spec. 5).

```
DataEnumeration enum_PostState values ("Draft", "Public")

DataEntity e_Blog "Blog": Parameter [
  attribute Name: String (30) [ constraints (NotNull Unique)]
  attribute Slogan: String (80) [ constraints (NotNull)]]

DataEntity e_Category "Category":  Reference [
  attribute Name: String (30) [ constraints (NotNull Unique]
  constraints (showAs (Name))
  description "Blog Categories"]

DataEntity e_User "MyTinyBlog User": Master [
attribute username: String [ constraints (NotNull Unique) tag (name "username" value
"username")]
  attribute firstName: String
```

```
  attribute lastName: String
  attribute email: Email [ constraints (NotNull) tag (name "email" value "email")]
  attribute registrationDate: Datetime [ defaultValue "CurrentDateTime" constraints
(NotNull ReadOnly) ]
  attribute bio: Text
  tag (name "user" value "user")]


DataEntity e_Post "Blog Post": Document: Regular [
  attribute state: DataEnumeration enum_PostState
  attribute title: String (30) [ constraints (NotNull)]
  attribute Body: Text [ constraints (NotNull)]
  attribute date: Datetime [ defaultValue "CurrentDateTime" constraints (NotNull ReadOnly) ]
  attribute category: String [ constraints (NotNull ForeignKey (e_Category))]
  attribute author: String [ constraints (NotNull ForeignKey (e_User))]


]

DataEntity e_Comment "Comments": Document: Weak [
  attribute post "Post ID": Integer [ constraints (NotNull ForeignKey (e_Post))]
  attribute comment "Comment": Text [ constraints (NotNull)]
  attribute date "Comment Date": Datetime [ defaultValue "CurrentDateTime" constraints
(NotNull ReadOnly)]
  attribute author "Post Author": String [constraints (NotNull  ForeignKey (e_User))]


]

DataEntity e_Like "Like":  Document: Weak [
  attribute like: Boolean
  attribute post "Post ID": Integer [ constraints (NotNull ForeignKey (e_Post))]
  attribute date "Comment Date": Datetime [ defaultValue "CurrentDateTime" constraints
(NotNull ReadOnly)]
  attribute user: String [ constraints ( NotNull ForeignKey (e_User)) ]
]
```

Spec 5. ASL Specification of MyTinyBlog data entities

After defining the data entities, **DataEntityClusters** can be defined. A **DataEntityCluster** construct denotes a cohesive set of structural entities that present logical arrangements among them and are commonly used in the context of use cases.

In this example, we define three data clusters with specific roles to their involved data entities. The “main” role represents the primary data entity involved, while the “child” role represents a “part of” (or “child”) data entity, and the “uses” role represents other logical dependencies between entities [9]. Furthermore, the tag “Inline” with value “Stacked” (see Table 5), in the ec_Post cluster, is used as an extended property (in what respect the UI definition of the application when it to comes to posts and comments).

```
//Application
DataEntityCluster ec_Blog "Blog": Parameter [main e_Blog]

//List View Post
DataEntityCluster ec_PostList: Document [main e_Post uses e_User, e_Category]

//Detail View Post
DataEntityCluster ec_Post: Document [main e_Post child e_Comment uses e_Category
  tag (name "Inline" value "Stacked") ]
```

Spec 6. ASL Specification of MyTinyBlog Data Entity Clusters

4.3. Use Cases

A use case is defined as a sequence of interactions between an actor(s) and the system under consideration, which gives some value to the actor [8,71]. Use cases is a popular technique of modelling user tasks, that can be complemented with informal storyboards and free-form scenarios [9]. Likewise with the RSL, ASL includes the UseCase construct that allows to define several properties such as: the involved DataEntity/DataEntityCluster; the actor that initiates the use-case and other participating actors or the actions that may be performed in the use case scope, e.g. CRUD actions.

In the MyTinyBlog (MTB) example (see Spec. 7), we define the ContextActor “Blog Editor” that creates and manages blog posts. The use case “Manage Blog Posts” is initiated by the “Blog Editor” and has actions over the data cluster “Blog Posts” (ec_Post) with CRUD actions. The only non-CRUD action is the action “aShare”. This action means users can share posts in social media and we explain in section 4.5 how this action can be defined.

```
ContextActor aU_Admin "Blog Administrator": User
ContextActor aU_Editor "Blog Editor": User
ContextActor aU_Reader "Blog Reader": User

ActionType aShare

UseCase uc_1_ManageUsers "Manage Blog Users": EntitiesManage [
  actorInitiates aU_Admin
  dataEntity e_User //this user should be able to create groups and permissions aswell
  actions aCreate, aRead, aDelete, aUpdate ]

UseCase uc_2_ManagePosts "Manage Blog Posts": EntitiesManage [
  actorInitiates aU_Editor
  dataEntity ec_Post //post with comments
  actions aCreate, aRead, aDelete, aUpdate, aValidate  ]

UseCase uc_3_ManageCategories "Manage Posts Categories": EntitiesManage [
  actorInitiates aU_Admin
  dataEntity e_Category
  actions aCreate, aRead, aDelete, aUpdate]

UseCase uc_4_BrowsePosts "Browse Blog Posts": EntitiesBrowse [
  actorInitiates aU_Reader
  dataEntity ec_PostList
  actions aRead, aShare ]

UseCase uc_4_1_CreateComment "Create Comment on Post" : EntitiesManage [
  actorInitiates aU_Reader
  dataEntity e_Comment
  actions aCreate, aRead, aUpdate]

UseCase uc_4_2_LikeComment "Like Comment on Post" : EntitiesManage [
  actorInitiates aU_Reader
  dataEntity e_Like
  actions aCreate  ]
```

Spec 7. Specification of Context Actors and Use Cases (in ASL)

4.4. User Interface Elements

As seen above, using ASL we can define Data Entities, Data Entities Clusters, Use Cases, Context Actors, and other constructs needed to specify the application. We may also define UI elements, namely (and following the IFML terminology): UI containers, UI components and UI parts. The rules to

express such elements in ASL are aligned with the IFML definition. The UI components supported by ASL are of the following types: List, Details, Form, Dialog and Menu. These UI components can be further classified as different sub-types like List-MultiChoice, List-Tree, List-Table, etc. as suggested in the next Tables.

Table 6 - Supported types for UI Containers in ASL

Type	**Window**	**Menu**	**Other**
Sub-Type	Modal	Main	Other
	Modeless	Contextual	
	Other	Other	

Table 7 - Supported types for UI Parts in ASL

Type	**Field**	**Slot**	**Other**
Sub-Type	Output	Menu Group	Other
	Input	Menu Group	
	Selection	Menu Separator	
	Editable Selection	Other	

Table 8 - Supported types for UI Components in ASL

Type	**List**	**Detail**	**Form**	**Dialog**	**Menu**
Sub-Type	MultiChoice		Simple	Success	Main
	Tree		MasterDetail	Error	Contextual
	Table		Other	Warning	
	Nested			Info	
				Message	

```
UIContainer uiCt_Delete_Warning : Window : Modal [

   component uiCo_DeleteWarning : Dialog : Dialog_Warning  [

part uip_Delete_Warning_Text: Field: Field_Output [Text defaultValue "Are you sure you want to
delete e_Post?"]
     event uiev_delete : Submit: Submit_Delete [navigationFlowTo uiCt_e_Post]
     event uiev_cancel : Submit: Submit_Cancel [navigationFlowTo uiCt_e_Post]

     ]
]
```

Spec 8. Example of Delete Interface (in ASL)

4.5. ASL Extensibility

We have demonstrated how ASL allows a very detailed description of what described applications should be at the final state. DataEntity attributes types are vast, from Strings, Integers to Email or Images. ASL also provides a way of customizing the types accepted by the ITLingo Studio, where ASL specs are written. A new DataAttributeType can be declared and it will be recognized and even suggested during the specification of the “System” where it was introduced.

This does not mean the DataAttributeType will have a corresponding attribute type for the desired framework files. However, it still is an indicator that we are referring to a different type attribute in our application specification. Using customized attribute types may be decisive in specific situations and the ASL can embrace these new types in a later version, according to how often they are used.

This is also possible with Use Cases action types. ASL users can introduce customized actions for use cases. Although they are not initially expected by the grammar parser, they can still have an impact, in a model-to-code transformation, if declared properly. We are going to look further into this impact in Chapter 5.

```
DataAttributeType GeoPoint [description "GeoPoint attribute type (for geospatial data)"]
ActionType aShare “Share blog posts”
```

Spec 9. Introducing a new Data Attribute Type and a new Action Type for ASL Use Cases

5. ASL Based Approach and Key Transformations

This chapter introduces and explains the ASL approach and its model-to-model (M2M) and model-to-code (M2C) transformations. This discussion is once again supported by the MTB application, briefly introduced in Chapter 4.

Figure 7 suggests the ASL approach proposed to systematically and rigorously define software applications based on the ASL language and with the possibility of automatically generating the software application for a specific software platform.

The proposed approach consists in 6 main tasks, as suggested by the Figure 7. Task 1 starts with a developer specifying the application's data and use cases models in the ASL. Task 2 automatically validates the quality of the partial model, if it is valid, the ASL tool support may run model-to-model transformations to generate ASL UI specifications (Task 3). Then, in the Task 4, the developer can still update and extend the generated model with their preferences and repeat the process (this is not illustrated in the figure for the sake of legibility). After this hybrid set of manual and automatic tasks, the complete model shall be validated (Task 5) before running model-to-code transformations (Task 6) and producing the source code artifacts for the target software infrastructure.

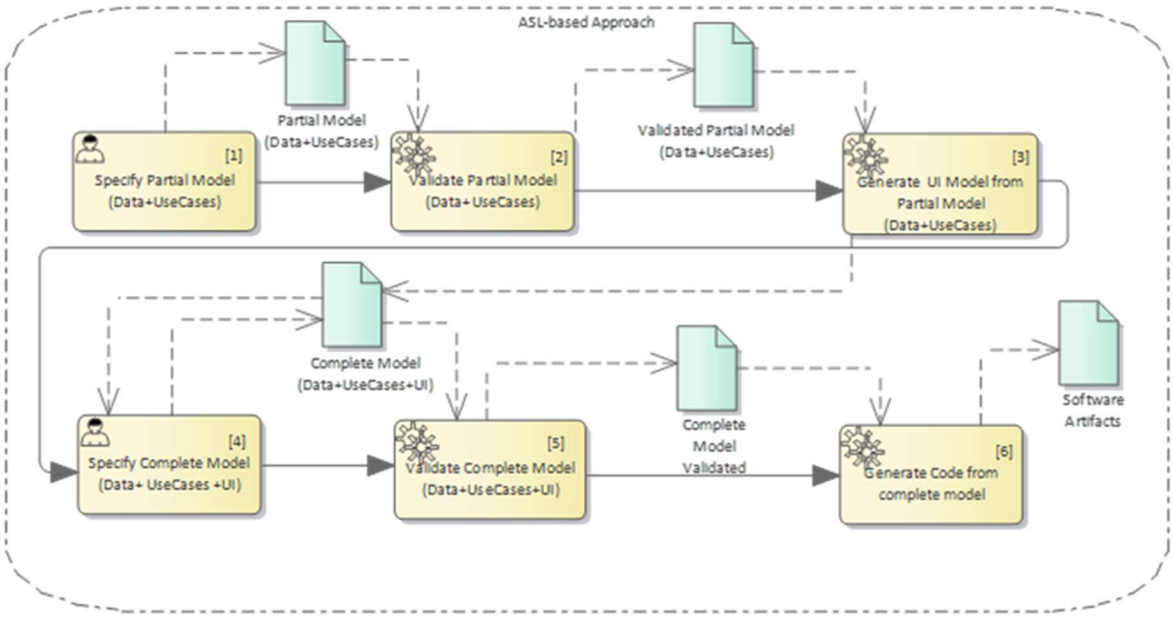

Figure 7 - ASL Based approach (in BPMN notation)

Due to their relevance, Task 3 and Task 6 will be described with more details respectively in sections 5.1 and 5.2.

5.1. Model-to-model transformation

The proposed approach follows an idea initially introduced with the XIS approach [47]: the idea of smart and dummy modeling approaches. According to that approach, the designer has just to define the Domain, Business Entities, Actors and Use Cases views (according to the XIS terminology), and then the User Interfaces views are generated based on model-to-model (M2M) transformations and a predefined set of UI patterns [47].

We integrate that "smart approach" in the ASL approach, which involves the generation of UI specifications, as referred in Task 3 of Figure 7. These generated ASL files include UI specifications that depend on the data entities and use cases previously defined. These files are generated through a Xtend file executed in the scope of the ITLingo Studio.

```
[…]
class ASLMainContainers {
  def static compile(UseCase usecase, ArrayList<UseCase> usecases, System system){

    var List<String> all_editable_attributes = newArrayList
    var List<String> readonly_attributes = newArrayList
[…]

'''
Package p_«system.name»

Import p_«system.name».«system.name»
Import p_«system.name».«system.name».*

System UI_«system.name» : Application [ ]

//UIContainer aligned with «usecase.name»
//MAIN WINDOW UI

UIContainer uiCt_«main_ent» : Window [
  component uiCo_«main_ent»List : List: List_Table [
    isScrollable

[…]
```

Spec 10. ASL UI Generator based on use-cases (Xtend file)

For each use-case a new .asl file is generated containing descriptions of interfaces for CRUD actions related to the target DataEntity or DataEntityCluster in the use-case.

For instance, for MyTinyBlog, 6 different .asl files were generated, each one with the name of the corresponding use-case (e.g. uc_2_ManagePosts.asl), as showed in Figure 8. These files have different elements regarding the actions in the use-case, but they follow a pattern and similar interfaces may be found in different files due to repeated actions over the same clusters.

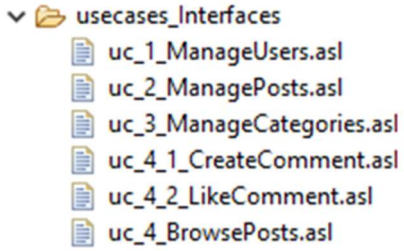

Figure 8 - User Interfaces generated for MTB

Considering the use-case defined in Spec.7 (i.e., use case "Manage Blog Posts"(uc_2_ManagePost)), a model-to-model transformation generates UI elements to support all CRUD actions of posts. Spec 11 shows part of the generated file.

```
UIContainer uiCt_e_Post: Window [

  component uiCo_e_PostList: List: List_Table [
    isScrollable
    dataBinding e_Post [
      visualizationAttributes e_Post.title, e_Post.category, e_Post.date
      sortAttributes e_Post.title, e_Post.category, e_Post.date
      orderBy e_Post.date DESC
  ]
  [...]
```

Spec. 11 - ASL List Component

The main components for this “UIContainer” in MyTinyBlog fit the purpose of listing posts, corresponding to the action “aRead” in “uc_2_ManagePost”. By default, a component of the type/subtype “List/ ListTable” was generated and the “databinding” mentions the **e_Post** entity, since this is the **main** entity for the **ec_Post** cluster. In this component we can customize the visualization, the sorting and the order in which instances of e_Post should be presented in this List.

Other features like listing, filtering and searching instances of features are also considered, since they are common or essential in business applications. These features can be manually customized in the .asl files, once again by choosing attributes of the target data entities referred in the use-case. The ITLingo-Studio tool gives suggestions when typing these modifications and will warn of errors (e.g. wrong attribute declaration) in case they exist.

Spec. 12 shows in another part of this generated .asl file, generic components for “reading blog posts” and how we can customize them.

```
  component uiCo_Filter_e_Post: Details [
    dataBinding e_Post [
      filterAttributes, e_Post.state, e_Post.author, e_Post.date
    ]
  ]

  component uiCo_Search_e_Post: Details [
    dataBinding e_Post [
      searchAttributes, e_Post.author, e_Post.title
    ]
  ]
```

Spec. 12 – ASL Search and Filter components

The complete specifications of these UI’s are available in the full version of this file, in Appendix A. Even though not all of them currently influence the M2C transformation, they are useful in representing and communicating features for web applications, regardless of the target framework.

Figure 9 shows an example of such a UI model (in IFML, on the top) that describes the search feature and the flow of data to the list of posts, and the corresponding UI of the final application developed in Django (on the bottom of the figure).

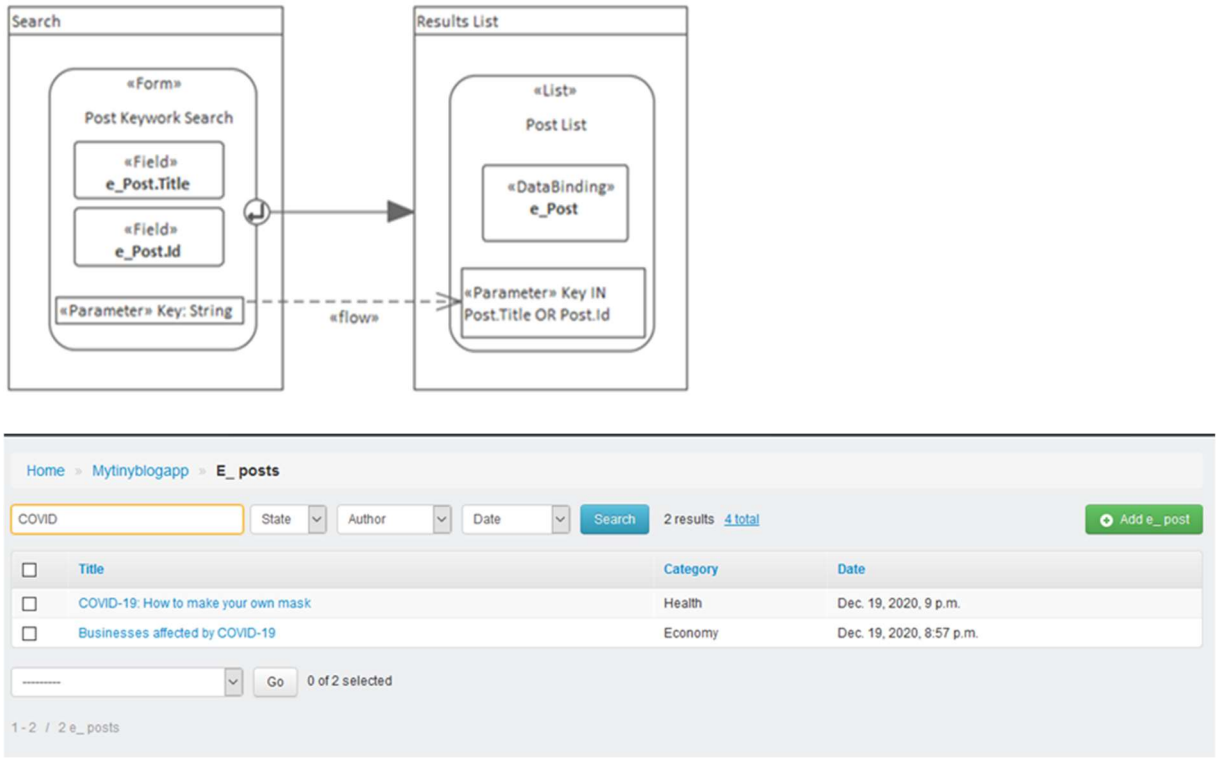

Figure 9 - Search posts: IFML model (top) and UI (bottom)

5.2. Model-to-code Transformation

The mode-to-model (M2M) transformation referred in section 5.1 (Task-3 in Figure 7) generates ASL UI specifications, but the target software application (e.g., MyTinyBlog) is not yet developed and deployed. However, the complete specification of the application under consideration can be used to produce the target application into a different number of software frameworks. As a proof of concept, we develop model-to-code (M2C) transformations for the Django web framework due to it being a framework that encourages rapid development and clean and pragmatic design [9,41]. From the MyTinyBlog specifications and the UI specifications generated from it, ITLingo Studio generates new files. Figure 10 shows a common output files tree when applying this transformation:

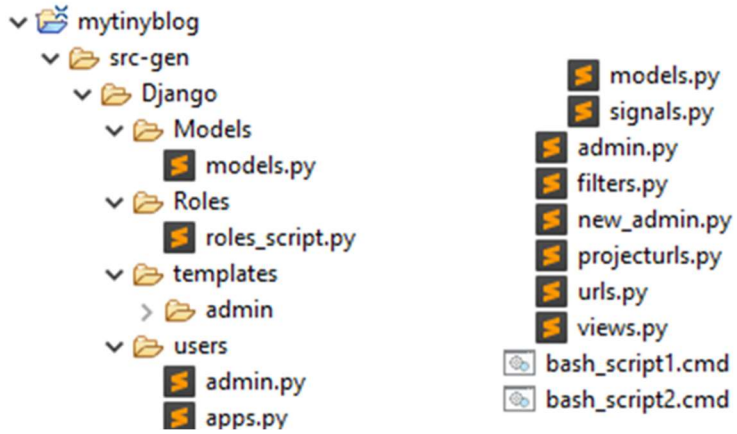

Figure 10 - M2C transformation output files

As showed in Figure 10, this set of files includes:

- A models.py script responsible for generating migrations to a database;
- A python script to generate groups, users and assigned permissions to groups;
- A basic template that customizes part of Django Admin;
- A “users” folder which associates Django users with a different and flexible model (Profile);
- A simple admin.py file that registers the use-cases entities to be used in Django Admin;
- An alternative admin.py that with more customized interfaces;
- A filters.py script based on the desired filters;
- 2 Generic urls.py files to be used at different levels (project and application), which are mandatory in Django projects;
- A views file that contains Class Based Views, the logic part of Django applications;
- 2 bash scripts that support the organization and creation of the Django project;

To use and speed up the process of deploying the MTB application with the above components, there is an “Deployment Instructions” document that explains how to make use of the bash scripts for Windows along with the generated Django application files. The web application is ready in a few minutes and the developers can proceed to make changes or test the application running it in localhost.

The main requirements for this deployment are having Python and Django installed along with a few additional packages: (i) a theme for Django admin interface, Django Suit; and (ii) Django-Filter, a Django application which simplifies filters definitions for applications. Other front-end frameworks and templates can be used to display data and logic provided by ASL. Spec. 6 illustrates the corresponding Django data model for the MTB application, produced from the model-to-code transformation.

```
#GENERATED ENTITIES
from django.db import models
from datetime import datetime
from django.contrib.auth.models import User
from django.core.exceptions import ValidationError
from django.core.validators import RegexValidator
#MIGHT BE NEEDED from django.contrib.gis.db import models

ENUM_POSTSTATE_CHOICES = ( ('draft','Draft'),  ('public','Public'), )

class e_Blog(models.Model):

   Name = models.CharField(max_length=20)

   Slogan = models.CharField(max_length=20)

class e_Category(models.Model):

   Name = models.CharField(max_length=20)

class e_Post(models.Model):

   state = models.CharField(max_length=15, choices=ENUM_POSTSTATE_CHOICES)

   title = models.CharField(max_length=20)

   Body = models.TextField()

   date = models.DateTimeField(default=datetime.now, blank=True)
```

```
   category = models.ForeignKey('e_Category', on_delete=models.CASCADE,
related_name='e_Post_category')

   author = models.ForeignKey(User, on_delete=models.CASCADE, related_name='e_Post_author')

class e_Comment(models.Model):

   post = models.ForeignKey('e_Post', on_delete=models.CASCADE, related_name='e_Comment_post')

   comment = models.TextField()

   date = models.DateTimeField(default=datetime.now, blank=True)

   author = models.ForeignKey(User, on_delete=models.CASCADE, related_name='e_Comment_author')

class e_Like(models.Model):

   like = models.BooleanField()

   post = models.ForeignKey('e_Post', on_delete=models.CASCADE, related_name='e_Like_post')

   date = models.DateTimeField(default=datetime.now, blank=True)

   user = models.ForeignKey(User, on_delete=models.CASCADE, related_name='e_Like_user')
```

Spec 13. Django model generated for MTB (in Python)

This code is generated mainly from the data entities defined in ASL (see Spec 5). This transformation generates the file “models.py”, which considers all the data entities, attributes, and data constraints, including foreign keys constraints.

This generated Python file defines the domain model with the application’s data structure and allows to create and update the respective database. Then, a developer can customize and refine that model and eventually add more information (by default, Django uses SQLite database to store the data [41]).

Users (who were given permissions) can create, read, update, or delete the blog posts using the Django admin site [41]. This site reads metadata from the models and provides a simple, yet efficient model-centric interface.

To perform CRUD operations, we need to register those models in the “admin.py” file. This file is created by default when a Django project is started. However, we have replaced it with new settings; It shall contain the models to be registered, and other constraints generated from the ASL specifications files (specs 11 and 12).

```
[…]
admin.site.register(e_Comment)

class e_CommentsInstanceInline(admin.StackedInline):
   model = e_Comment
   extra=0
   filter=()
   readonly_fields=('date', )

@admin.register(e_Post)
class e_PostAdmin(admin.ModelAdmin):
   inlines = [e_CommentsInstanceInline,  ]
   exclude = ()
   readonly_fields=('date', )
[…]
```

Spec 14. Part of the admin.py file generated by ASL reflecting the relation main-child between posts and comments (in Python)

The name of the app which can be important for some businesses and the application users is visible in Django Admin through modifications in templates like the following:

```
{% extends "admin/base.html" %}

{% block title %}{{ title }} | MyTinyBlog{% endblock %}

{% block branding %}
<h1 id="site-name"><a href="{% url 'admin:index' %}">MyTinyBlog</a></h1>
{% endblock %}

{% block nav-global %}{% endblock %}
```

Spec 15. Template generated in ASL for MTB

For sakes of legibility and styles, we have been using a theme for Django admin interface, Django Suit [51]. This is a view of posts list considering the search and filters fields in admin.py files:

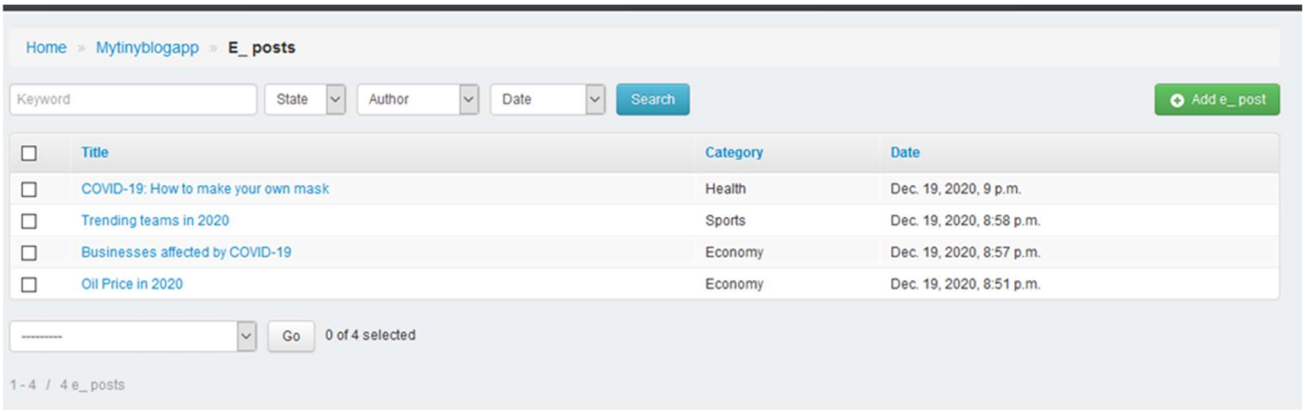

Figure 11 - MyTinyBlog posts list

5.2.1. Django Views

One of the three parts of the MTV Django architecture are Views, “Python callables that accept an HttpRequest object as argument and returns an HttpResponse object is deemed a view in Django” [41]. In Django we have three different types of views: function views (FV), class-based views (CBV) and generic class-based views (GCBV) [41, 52].

A CBV is any class that inherits from View. The Difference between FV and CBV is that the last one has several benefits, among them automatic handling of HTTP options, instead of branching code with conditions [52].

In its turn, GCBV is a CBV that comes to address the common use cases in a web application, such as creating or listing objects, form handling, pagination, etc. They make use of attributes that must be configured, which can be one of the disadvantages of working with CBV: trying to work out exactly which and how pre-programmed methods need to be configured [52].

```
def BlogPostListView(request):
    posts = e_Posts.objects.all()

     […]

    context={
        'queryset': posts,
        […]
```

```
    }

    return render(request, "mytinyblog/e_Posts_list.html", context)
```

Spec 16. Example of a function view in Django (list of Blog Posts)

```
class BlogPostListView(ListView):
    model = e_Post
    context_object_name = 'posts'
    template_name = 'mytinyblog/e_Posts_list.html'

[…]
```

Spec 17. Example of a Django Generic Class-Based View (ListView)

For the views, we will be using Django Generic Class Based Views. For each “CRUD” action, we need a corresponding view: CreateView, ListView, UpdateView, DeleteView.

In our MyTinyBlog example, there is at least one actor, that can create posts, therefore, a CreateView will be generated. “Blog Editors” can create “Blog Posts”, therefore they are given such permissions. In this view, we make sure only “Blog Editors” can access it but the same assertion could be done by testing if the actor was in “Blog Editors” group.

```
[…]
class e_PostCreateView(LoginRequiredMixin,UserPassesTestMixin, CreateView):
  model = e_Post
  #form_fields = []
  #success_url = reverse_lazy('e_Post-list')

  def test_func(self):
    if self.request.user.groups.filter(name='aU_Editor').exists():
      return True
    else:
      return False
[…]
```

Spec 18. CreateView generated in ASL for MyTinyBlog (form_fields and success_url need to be revisited by the developer to assure they work as expected in their application)

5.2.2. Filters

Filter options are also taken in consideration for MTB front-end. They are relevant for ListViews, in order to help users accessing the objects in a long list. This is the generated “filters.py” file that can be connected to a template. In this file we write what fields can be used to filter posts. We use the application “django_filters” [53], that simplifies the creation and usage of such filters.

```
class e_PostFilter(django_filters.FilterSet):
  class Meta:
    model = e_Post
    fields = ['author', 'category', 'title',]
```

Spec 19. Generated Filters.py that contains fields to filter MyTinyBlog posts

5.2.3. Users, Groups and Permissions

The implemented model-to-code transformations can speed up the process of managing users and permissions (see Table 2 for some concepts mapping between ASL and Django. We have seen that Django Admin provides a built-in authentication system that allows to create groups and assign permissions to users [54]). Using Python interpreter, we can create groups and assign permissions to users through a Python script.

In the MyTinyBlog application, the blog administrator oversees all the tasks. In his turn, the blog editor should be able to create, read, update and delete posts. ASL tool generates a Python script to insert groups, users, and permissions in the database. To quickly validate the authorization features, this script adds one user to each group instance. All these settings can be later directly managed by a superuser using the Django admin site, or again the python interpreter.

Table 9 - ASL to Django concepts mapping

ASL	**Django**
ContextActor	Group Instance
ContextActor (name)	User
UseCase Actions	Permissions

```
from django.contrib.auth.models import Group
from django.contrib.auth.models import Permission
from django.contrib.auth.models import User

aU_Editor_group = Group(name='aU_Editor_group')
aU_Editor_group.save()

user = User.objects.create_user('aU_Editor', password='password')
user.is_staff=True
user.save()
aU_Blogger_group.user_set.add(user)

permission_CreatePost = Permission.objects.get(codename='add_e_post')
aU_Blogger_group.permissions.add(permission_CreatePost)
```

Spec 20. Part of the generated Django roles script for MTB (in Python)

Apart from other interfaces and functions being added, the Blog Administrator will be able to manage permissions, users, and groups in Django Admin:

Figure 12- Groups Page in Django Admin

The generated user application contains, among other important files, the model for a user profile with the fields chosen in ASL, when we used the tag with name and respective value “user” (spec. 5)

```
from django.db import models
from django.contrib.auth.models import User
from datetime import datetime

class Profile(models.Model):
  user = models.OneToOneField(User, on_delete=models.CASCADE)
  username = models.CharField(max_length=20)
  firstName = models.CharField(max_length=20)
  lastName = models.CharField(max_length=20)
  email = models.EmailField(max_length=254)
  registrationDate = models.DateTimeField(default=datetime.now, blank=True)
  bio = models.TextField()

  def __str__(self):
    return f'{self.user.username} profile'
```

Spec 21. Generated Django profile model for MyTinyBlog users (in Python).

Each profile has a one-to-one relation to the users of MyTinyBlog and in Django Admin we can edit them.

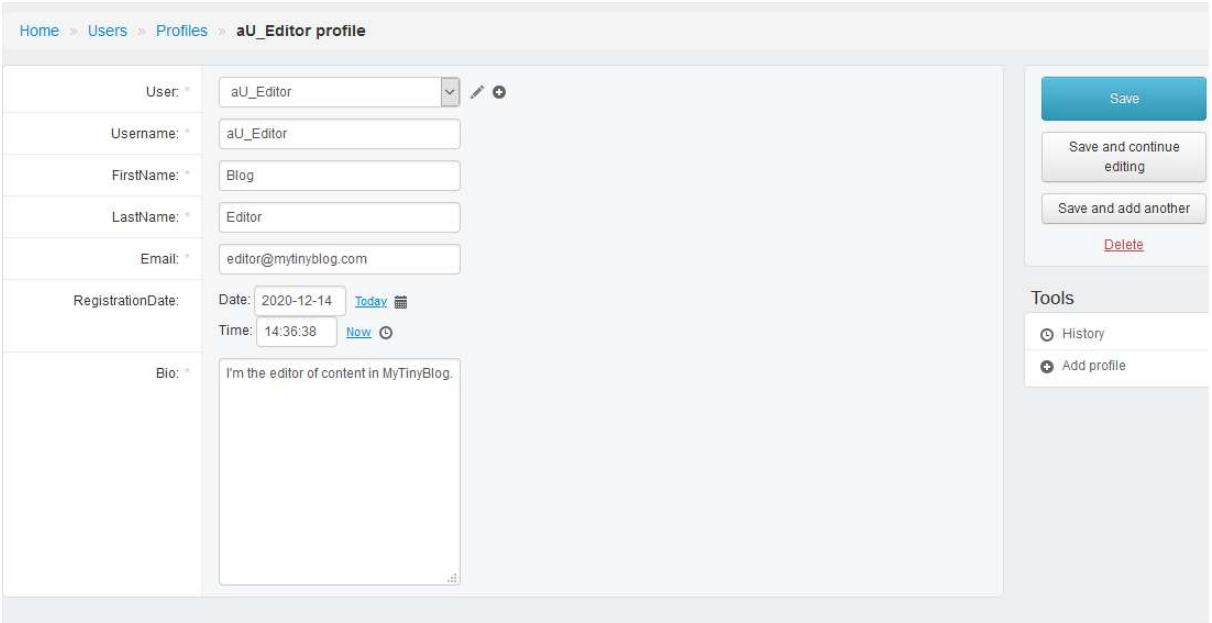

Figure 13 - User Profile Form in Django Admin

5.2.4. URLs

A front-end interface could be added now created for MyTinyBlog through templates, url paths and the views generated. During that process, developers can use the ASL UI concepts in ITLingo Studio or use another tool, like IFML in Enterprise Architect [55], in order to get the a more user-friendly and complete version of MyTinyBlog. The developers would need to add new urls.py configurations. Through ASL, the urls.py configurations that are responsible for connecting different areas of the blog are very basic:

```
from django.urls import path, include
from . import views

urlpatterns = [
  path('', views.home, name='home'),
  path('/about', views.about, name='about-page'),
]
```

Spec 22. Urls.py generated for MyTinyBlog application

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
  path('admin/', admin.site.urls),
  path('', include('MyTinyBlogApp.urls'))
]
```

Spec 23. Urls.py generated for MyTinyBlog (project level)

6. Validation

This chapter presents and analyses case studies where we applied the ASL approach described in Chapter 5.

6.1. Case Study A: the MyTinyBlog

The Case Study A was introduced in Chapter 4 to support the explanation of the ASL language and to show how to generate business applications through model-to-model and model-to-code transformations, as discussed in Chapter 5.

Blogs are popular case studies when introducing developers to new methodologies or programming languages and can often be enriched with new features and with different levels of complexity. Blogs can be used to show how to run unit tests, to deploy web applications, or to make asynchronous requests. In the scope of this research, the MyTinyBlog application helps us to introduce common practices to CRUD operations and different ways to display data in webpages.

6.1.1. Impact of the ASL approach in the development

In MTB, ASL was effective and efficient. The data entities and attributes are straightforwardly created. The groups were easily identified, and their roles defined. The UI components are enough for the blog usability, as Django Admin could now be used to perform the application main operations. It is important to notice that the views.py file is not being used in our demonstration.

We can analyze the complexity of our applications keeping track of the number of data entities, clusters, use-cases, actors, etc. in the ASL specification. We can follow these numbers with an analysis of lines of code were generated and their coverage level, regarding how many manual changes were likely to happen since the files were generated from ITLingo Studio to be used in another text editor or development environment, such as Visual Studio Code [56]. In MyTinyBlog we chose to use CLOC (Count Lines of Code) [58], a software that not only counts lines of source code in many programming languages (Python being one of them) but also blank lines and comment lines.

In MTB specifications, using ITLingo Studio, we had a system with approximately 100 lines of specification:

Table 10 - ASL Specification Analysis – Case Study A

ASL Element	**# itens**	**Lines of Specs**
Data Entities	6	46
Data Entity Clusters	3	6

Data Enumeration	1	1
Context Actors	3	3
Use-Cases	6	35
Custom Actions for use-cases	1	1
Other elements	-	6
		98

Generated in M2M transformation, even though not widely used and containing some repeated code, the use-cases interfaces contain: **650** lines of specs evenly divided in 6 files for each use-case. 2 of the 6 files had interfaces for the entity “e_Post” and contained repeated code (**110** lines of code each).

For the demonstration in section 5.1, we used **22** lines that allow the specification of search, filter and visualization fields of Posts in MyTinyBlog and impact the subsequent transformation (M2C). This corresponds to **20%** of the lines of code, regarding posts (110).

The M2C transformation for MTB resulted in a folder named “Django” and its structure is identical to the structure presented in Chapter 5 – Figure 10.

This is the analysis of the folder over the files that were generated in ITLingo Studio:

```
      12 text files.
      12 unique files.
       6 files ignored.

github.com/AlDanial/cloc v 1.88  T=0.09 s (141.2 files/s, 5482.4 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Python                          11            158             18            281
HTML                             1              3              0              6
-------------------------------------------------------------------------------
SUM:                            12            161             18            287
-------------------------------------------------------------------------------
```

Figure 14 - Analysis of the generated files (CLOC)

```
      20 text files.
      20 unique files.
      18 files ignored.

github.com/AlDanial/cloc v 1.88  T=0.09 s (208.2 files/s, 6355.8 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Python                          18            170             53            348
HTML                             1              3              0              6
-------------------------------------------------------------------------------
SUM:                            19            173             53            354
-------------------------------------------------------------------------------
```

Figure 15 - Analysis of deployed MyTinyBlog files (CLOC)

Table 11- MyTinyBlog lines of code statistics

MyTinyBlog				
ASL Specifications		Application		
Manual	Generated	Manual	Generated (CLOC)	Deployed (CLOC)
98	650	5	287	354

The code generation ratio through the ASL Approach is **287** (Application Generated (CLOC)) / **98** (Manual ASL Specifications) = **2.92** (almost triplicated)

After **5** lines manually changed and the deployment in a Windows machine, the MyTinyBlog has a total of **354** lines of code, so the coverage of the ASL Approach is **287/354** = **81%**

6.2. Case Study B: the RiverCure Portal

The second study case for the ASL approach is the RiverCure Portal (RCP) which main goal is to improve water resources management and habitat protection. By having a solid information system that collects all the useful data, technicians can reduce uncertainty and improve their hydrodynamic and morphodynamic mathematical models. A digital platform that could collect and maintain data from several sources (such as SVARH, HiSTAV and crowdsourcing) is necessary [13].

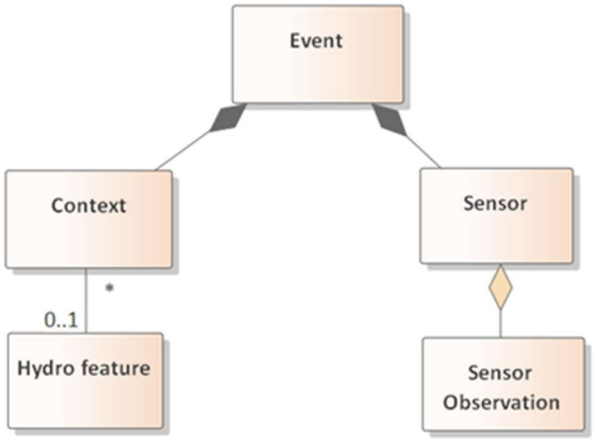

Figure 16 - RiverCure Portal data model (simplified)

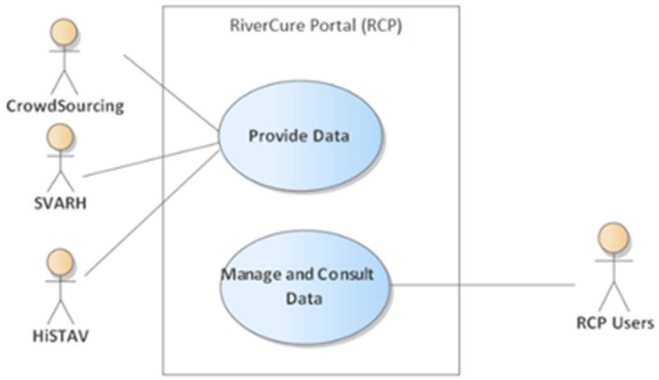

Figure 17 - RiverCure Portal Use Cases Diagram (simplified)

RCP is a Django web application and we can divide its flow of operations in 4 main modules**: general information**, in which **hydro features** are the most relevant data; **sensors** and respective **observations**; **contexts**, complex data entity clusters which main attributes are from the geographical domain (polylines and points) that should be defined by experts in order to have a purposeful meaning. Last, but not least, **events** that depend on the other 3 modules.

The first module contains some information that can be accessed by people that do not perform any core activities in the RCP, like hydro features, but the management of these elements and others, such as Cities or Parishes is done by a set of users in the group 'Managers'.

In the second module, the responsible users belong to the group 'Sensor Managers'. Currently, the sensors and respective observations data are not yet provided by any official external system. This module is thought to be integrated in the future with official information from SVARH (Sistema de Vigilância e Alerta de Recursos Hídricos). SVARH is a system that allows to know the hydrological state of Portuguese rivers and dams, like levels laid up volumes and quality of water. Along with that, it collects relevant meteorological information and allows a better understanding and prediction of the collected and future data [59]. SVARH is a system that is available to entities related to hydric resources management and it is essential in ANPC [60] missions which aim to save people and goods

upon dangerous events, such as floods [59, 61]. For now, sensor managers create sensors and respective observations using the RCP interface. They can introduce sensors locations by introducing coordinates or placing a marker in a map. Both sensors and observations can be uploaded to RCP in an Excel spreadsheet by a Python script that reads its information and updates the RCP PostgreSQL [62] database.

This module should be provided more data through the future integration of a web application (crowdsourcing in Figure 17) developed in Python by Jorge Pereira [63]. Upon its integration, all RCP users, including visitors shall have access to a photo gallery and should be able to submit their own photos. These photos provide information that includes metadata and an estimation of flood risk by measuring water heights [63].

RCP contexts are managed by Context Administrators and Context Managers. The first group of users can create a context from scratch and assign permissions to other Context Administrators or Managers related to the contexts they “own”. Contexts are constituted by meta-data such as name, code or hydro feature, and their geographical elements is not only their most important part but also the RCP most distinguished feature. Users have the option to upload information such as digital model terrain (DTM), domain, alignments, refinements, boundaries, and contour lines for each context. On the other hand, they can create and manipulate these elements in RCP itself [64]. After uploading/creating their context manually, they can associate existing sensors to the context boundary points and assert its validation through a button (e.g. one context must have a domain). Once it is validated, the context can be chosen to generate a mesh and to be target in events, such as simulation events through HiSTAV.

HiSTAV is a high-resolution simulation tool that uses a 2DH (two-dimensional horizontal) mathematical model based on shallow water equations featuring dynamic bed geometries and sediment transport to simulate fluvial and estuarine flows [13, 65]. This tool allows the forecasting of an overland tsunami and assists in the decision-making process regarding water-related hazards [13, 65].

The flow of operations in HiSTAV can be divided in 3 steps [13]: in the first step, a pre-processing tool prepares the data that is going to be applied in the model. The HiSTAV receives data inputs from RCP such as: initial conditions, boundary conditions, physical constants. Context Administrators and Managers can generate a mesh, from a context detail page when the necessary inputs are defined and validated. At this point, users who belong to the group Context Event Managers can procced to request a simulation in HiSTAV by creating an event for the context (2^{nd} step). The simulation request is sent through a virtual private network (VPN) that connects the RCP server and the HiSTAV [64]. A simulation can take a long time depending on the complexity of the data being processed and generates output files that can be post-processed by tools such as ParaView [66]. In the third step: the RCP receives the simulation results from HiSTAV and the user responsible for this simulation can access and interpret the simulation resulting files using the tool ParaView Web Visualizer [67].

The BPMN diagram displayed in Figure 18 represents the described activities from the moment a context is created and a simulation for it ends. Due to the existence of different types of users in RCP, we consider a “Super User” that can perform all the activities to simplify this representation:

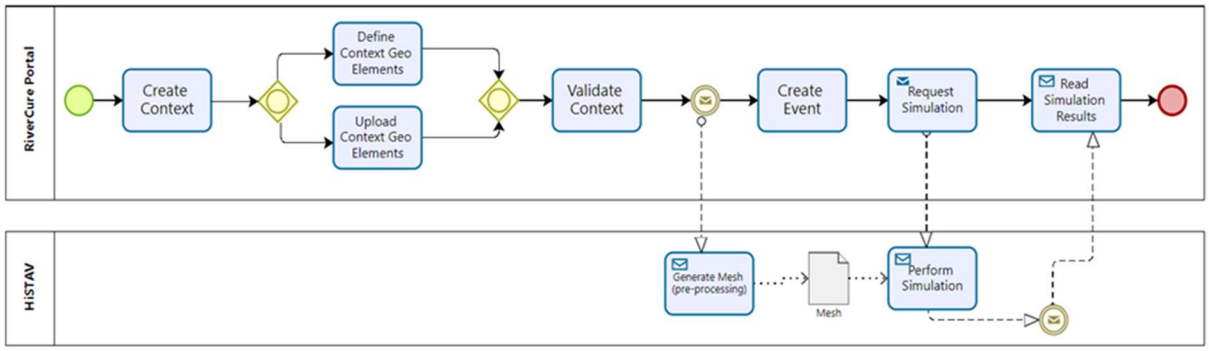

Figure 18 - Activities diagram of Context creation (in BPMN)

6.2.1. RiverCure Main Challenges

RiverCure Portal is distinct from all the previous study cases in ITLingo. It is very different from MyTinyBlog, but also the ‘BillingSystem’, the most iconic study case in RSL [11]. It contains lots of data entities and some of them have new data attributes types (e.g. geographical attributes).

There are several challenges in this project. One of the main ones is the existence of several and different stakeholders. These stakeholders have different knowledge, impacting the RiverCure Portal development in different ways. They come from different departments and areas, such as Civil Engineering and Information Systems.

The data provided by some sources can be a problem because currently standards are not well defined and used in the involved organizations. One example is the geographical coordinates for hydrometric sensors, one of the most important type of entities in this project.

Another challenge is the fact that data is needs to be sent and processed by external systems, such as HiSTAV and that increases the complexity of the development.

Some of the RiverCure use-case actions are not found in the previous study cases, which also leads to a richer set of permissions and groups of actors. This is observed in some of the permissions that are single object specific and not entity specific. Finally, the ending system can have a serious impact both in academic investigation and society, supporting how authorities can forecast and act upon natural events.

This is part of the ASL Specification for RiverCure Portal, based in a RSL specification previously developed by Marta Gonzalez [13] and that was target of improvements and refinements throughout the development of the web application, due to the complexity of the product being developed and the fruitful discussions that happened during the development with different stakeholders. The interoperability between the two languages (RSL and ASL) made the task of converting the system into ASL easier.

6.2.2. M2C Transformations for RCP

```
//Sensors: General Sensor, e_FixedInSituSensor, e_WeatherSensor, e_PhotoSensor

DataEnumeration SensorKind "Sensor Kind" values ( "HydrometricSensor", "WeatherSensor",
"SocialNetworkScanner", "HumanSensor", "TBD Sensor" )

DataEnumeration SensorModalityKind "Sensor Modality Kind" values ("PhysicalFixed",
"PhysicalMobile", "DigitalSocialNetworkScanner", "DigitalHumanUpload" )

DataEnumeration MetricKind "Metric" values ("Sec", "Min", "Hour", "Day", "Week", "Month",
"Year")

DataEntity e_Sensor "Sensor": Master [
  attribute id "Id": Integer [constraints (PrimaryKey)]
  attribute code "code": String [constraints (NotNull Unique)]
  attribute Name "name": String [constraints (NotNull)]
  attribute type "Type": DataEnumeration SensorKind [constraints (NotNull)]
  attribute modalityType "Modality Type": DataEnumeration SensorModalityKind [constraints
(NotNull)]
  attribute Description "Description": Text
  attribute Version "Version": String
  attribute responsible "User Responsible": Integer [constraints (NotNull ForeignKey(e_User))]
  attribute geom "Geometry": GeoPoint

// FixedInSituSensor
  attribute recRhythmValue "Recording rhythm value": Integer []
  attribute recRhythmMetric "Recording rhythm metric": DataEnumeration MetricKind

// HydrometricSensor
  attribute zeroLevelScale "Zero level scale": Decimal

// WeatherSensor
  attribute maxRange "Max range": Integer
  attribute PRF "Pulse repetition frequency (PRF)": Integer
  attribute recRhythm "Recording rhythm": String

// PhotoSensor
  attribute isSocialNetwork "is from a social network": Boolean [defaultValue "False"]
  attribute isUpload "is from an upload": Boolean [defaultValue "False"]
  attribute source "Source": String
  attribute url "URL": URL
  attribute isAgreeTerms "is agreed with terms and conditions": Boolean [defaultValue "True"]]
```

Spec 24 - ASL Specification for the data entity regarding sensors (e_Sensor) and respective attributes

```
SENSORKIND_CHOICES = (  ('HydrometricSensor','Hydrometric Sensor'),  ('WeatherSensor','Weather
Sensor'),  ('SocialNetworkScanner','Social Network Scanner'),  ('HumanSensor','Human Sensor'),
('TBDSensor','TBD Sensor'),  )

SENSORMODALITYKIND_CHOICES = (('PhysicalFixed', 'Physical Fixed'), ('PhysicalMobile',
'Physical Mobile'),  ('DigitalSocialNetworkScanner','Digital Social Network Scanner'),
('DigitalHumanUpload','Digital Human Upload'),  )

METRICKIND_CHOICES = (  ('sec','Sec'),  ('min','Min'),  ('hour','Hour'),  ('day','Day'),
('week','Week'), ('month','Month'), ('year','Year'),)

class e_Sensor(models.Model):

  code = models.CharField(max_length=20)

  Name = models.CharField(max_length=20)

  type = models.CharField(max_length=15, choices=SENSORKIND_CHOICES)

  modalityType = models.CharField(max_length=15, choices=SENSORMODALITYKIND_CHOICES)

  Description = models.TextField()

  Version = models.CharField(max_length=20)

  responsible = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='e_Sensor_responsible')

  geom = models.PointField()
```

```
  recRhythmValue = models.IntegerField()

  recRhythmMetric = models.CharField(max_length=15, choices=METRICKIND_CHOICES)

  zeroLevelScale = models.DecimalField(max_digits=5, decimal_places=2)

  maxRange = models.IntegerField()

  PRF = models.IntegerField()

  recRhythm = models.CharField(max_length=20)

  isSocialNetwork = models.BooleanField()

  isUpload = models.BooleanField()

  source = models.CharField(max_length=20)

  url = models.URLField(max_length=200)

  isAgreeTerms = models.BooleanField()
```

Spec 25 - Example of a final model in Django for the Sensors entities

This is an example of a specification for use-cases about sensors in ASL for RCP. There is not a single way to write use-cases and there could be shorter ways of specifying the same actions, but the amount of entities, clusters, use-cases and actors in this study case makes a more structured specification necessary. We chose to group some of the use-cases through extension points in order to improve the legibility and to make changes easier when it comes to the use-cases and involved entities.

```
DataEntityCluster ec_Sensor "Sensor Simple": Master [main e_Sensor]
DataEntityCluster ec_SensorComplete "Sensor": Master [main e_Sensor child e_SensorObservation]

ContextActor aU_SensorsManager "SensorsManager": User [description "manages sensors"]

UseCase uc_7_ManageSensors "Manage sensors": EntitiesBrowse [
  actorInitiates aU_SensorsManager
  dataEntity ec_Sensor
  actions aRead]

UseCase uc_7_1_ManageSensor "Manage a single sensor": EntitiesManage [
  actorInitiates aU_SensorsManager
  dataEntity ec_SensorComplete
  actions aCreate, aRead, aUpdate, aDelete
  extensionPoints xp_ManageSensorData]

UseCase uc_7_1_1_ManageSensorData "Manage Sensor Observations": EntitiesManage [
  actorInitiates aU_SensorsManager
  dataEntity e_SensorObservation
  actions aCreate, aRead, aUpdate, aDelete
  extends uc_7_1_ManageSensor onExtensionPoint uc_7_1_ManageSensor.xp_ManageSensorData]
```

Spec 26. Actors and Use Cases (regarding sensors in RCP)

```
from django.contrib.auth.models import Group
from django.contrib.auth.models import Permission
from django.contrib.auth.models import User

aU_SensorsManager_group = Group(name='aU_SensorsManager_group')
aU_SensorsManager_group.save()

user = User.objects.create_user('aU_SensorsManager', password='password')
user.is_staff=True
user.save()
aU_SensorsManager_group.user_set.add(user)

permission_aRead = Permission.objects.get(codename='view_e_sensor')

aU_SensorsManager_group.permissions.add(permission_aRead)
```

```
permission_aCreate = Permission.objects.get(codename='add_e_sensor')
permission_aRead = Permission.objects.get(codename='view_e_sensor')
permission_aUpdate = Permission.objects.get(codename='change_e_sensor')
permission_aDelete = Permission.objects.get(codename='delete_e_sensor')

aU_SensorsManager_group.permissions.add(permission_aCreate)
aU_SensorsManager_group.permissions.add(permission_aRead)
aU_SensorsManager_group.permissions.add(permission_aUpdate)
aU_SensorsManager_group.permissions.add(permission_aDelete)

permission_aCreate = Permission.objects.get(codename='add_e_sensorobservation')
permission_aRead = Permission.objects.get(codename='view_e_sensorobservation')
permission_aUpdate = Permission.objects.get(codename='change_e_sensorobservation')
permission_aDelete = Permission.objects.get(codename='delete_e_sensorobservation')

aU_SensorsManager_group.permissions.add(permission_aCreate)
aU_SensorsManager_group.permissions.add(permission_aRead)
aU_SensorsManager_group.permissions.add(permission_aUpdate)
aU_SensorsManager_group.permissions.add(permission_aDelete)
```

Spec 27. Script to create groups and assign permissions (regarding sensors in RCP)

Trough Spec 27., we can notice that our organization of use-cases specification generated a duplicated permission (‘view_e_sensor ‘) in our script, but that will not cause any undesired effect when we run this script.

RCP pages regarding sensors

The RCP pages represented in the next Figures are accessible to users who belong to the group ‘SensorManager’. Users who don’t belong to this group will not be able to perform most of the actions here displayed and the interface will be properly adjusted to them (visitors can’t access these pages at all).

This is the main page of the sensors’ module. From here, a sensor manager can consult information regarding sensors registered in RCP, search, filter or order them by certain fields. They also can access individual sensors, its observations, add or upload a new sensor.

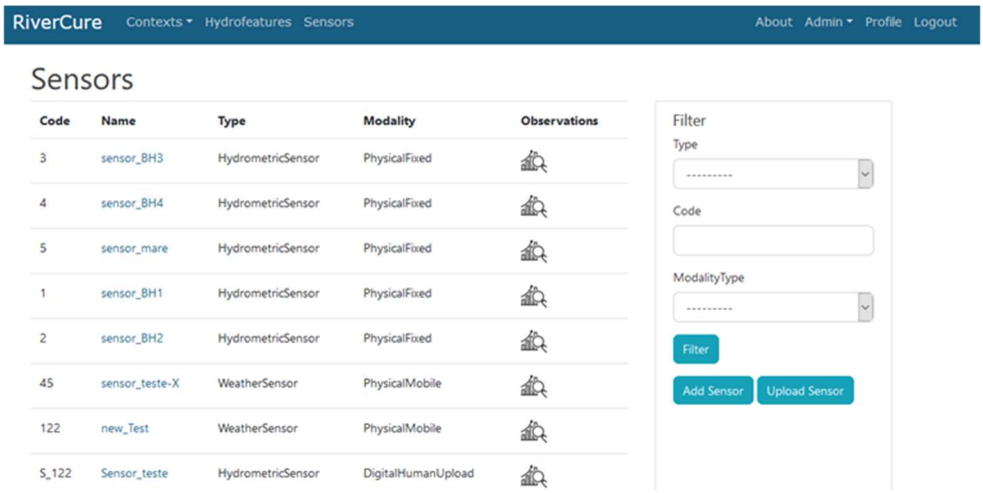

Figure 19 - Sensors List in RiverCure Portal

Accessing a sensor detail page, sensor managers they can access more information and perform new operations, such as editing or deleting the sensor.

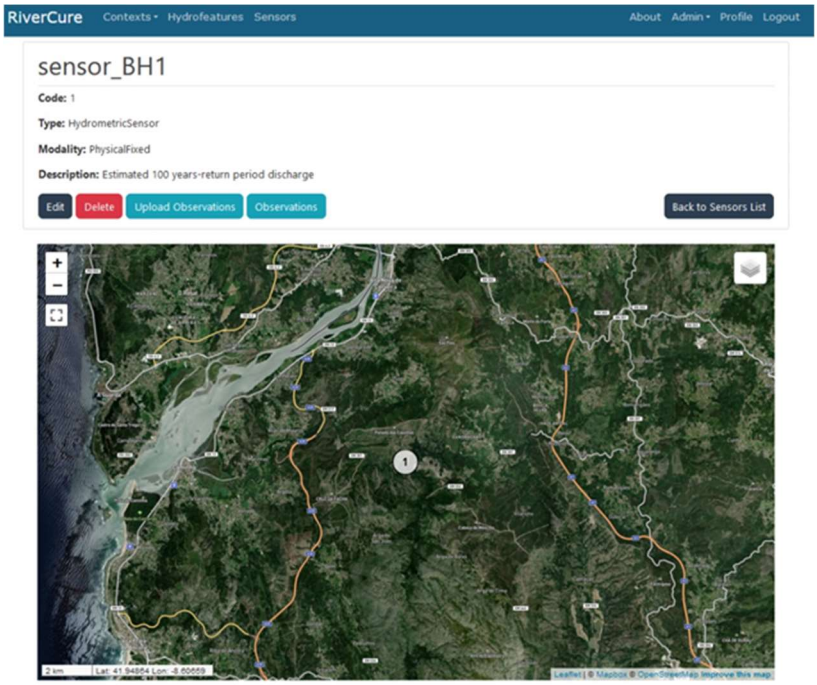

Figure 20 - Sensors detail page in RiverCure Portal

Using the button with the label “Observations” we navigate to a page that contains a list table with observations that belong to this sensor:

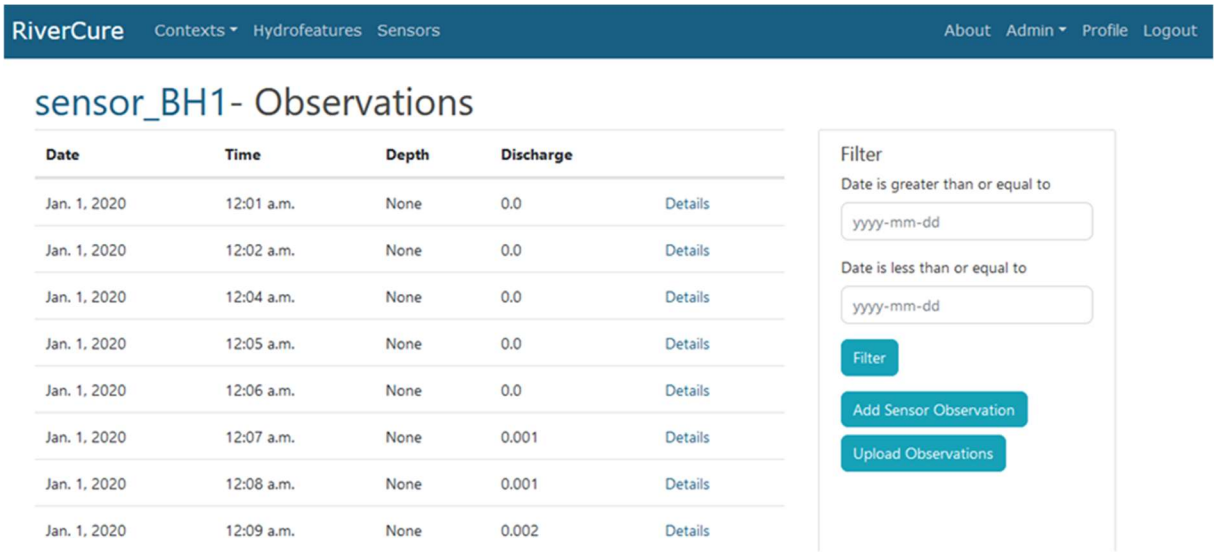

Figure 21 - Sensor Observations Page in RiverCure Portal

Similarly, to the sensors, each sensor observation has a page with detailed information:

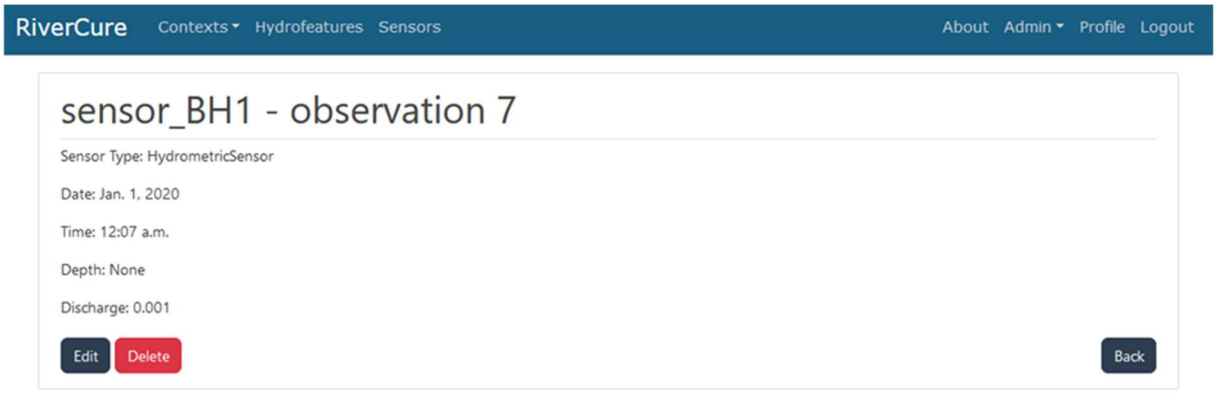

Figure 22 - Sensor observations detail page in RiverCure Portal

6.2.3. Impact of the ASL approach in the development

In RCP, the ASL approach had the most impact in the data model's definition. It also helped identifying different groups of users and limiting their access to perform actions in different modules through a python script (Spec. 27).

On the other hand, the impact it had defining the logic and presentation layers (views, urls, templates) was minimal because most of the actions over a single object was often related to other objects and parts of the application (e.g. relations between contexts and sensors), or even other systems, like HiSTAV. The Django class-based views did not cover the necessary actions to define geographical elements (e.g. context domains, boundaries, etc.) or uploading data from files (e.g. sensors' observations). Django CBV ended up being used at some cases, but it was often necessary to customize them following a standard practice of code production by the developers. Nevertheless, when it came to asserting the permissions for views and templates, the use-cases in ASL were an important source of information. Even though, data entities were registered in order to be used in the Django Admin interface, it is not a central part for this project after it has been deployed, except for administrators or managers when accessing certain entities that are not available in the RCP front-end, like Cities or Parishes, which makes the admin.py files less relevant for most of the RCP users.

Once again, using CLOC, we can check how many lines of Python code were generated in the model-to-code transformation. The RCP specification in ITLingo Studio suffered lots of modifications during this project and usually had a higher number of lines (that made it easy to work with and interact with stakeholders when necessary), but in order to display the actual M2M and M2C transformations, we had a .asl file with approximately 520 lines.

Table 12 - ASL Specification Analysis – Case Study B

ASL Element	# itens	Lines of Code
Data Entities	19	213
Data Entity Clusters	9	9
Data Enumeration	19	19

Context Actors	7	7
Use-Cases	19	117
Data Attribute Types	4	4
Other elements	-	21
Total		**390**

Generated in our M2M transformation, even though not widely used and containing some repeated code, the use-cases interfaces contain 2090 lines of code evenly divided in 19 files for each use-case.

We did not use these components for specification of searching, filtering and visualizing fields like we did for MyTinyBlog.

The M2C transformation for RiverCure Portal resulted in a folder named “Django” and its structure is identical to the structure presented in Chapter 5 – Figure 10. The biggest influence of ASL in RCP is in: (i) the models; (ii) scripts for groups/permissions; (iii) admin.py files (modified); (iv) users’ folder. We analyze these files with the tool CLOC after the M2C transformation in ITLingo Studio and then compare it to a recent version of the RCP that has been deployed.

This is the analysis of the folder over the files that were generated in ITLingo Studio:

```
github.com/AlDanial/cloc v 1.88  T=0.09 s (163.3 files/s, 23184.8 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Python                          13            875             87           1017
HTML                             1              3              0              6
-------------------------------------------------------------------------------
SUM:                            14            878             87           1023
-------------------------------------------------------------------------------
```

Figure 23 - Figure 14 - Analysis of the generated files for RCP (CLOC)

```
github.com/AlDanial/cloc v 1.88  T=0.76 s (161.3 files/s, 24370.4 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
CSS                              5           1495             14           9040
Python                          61            856            233           3051
HTML                            45            425              8           1945
JavaScript                       3             82            134           1116
Sass                             6             18              1            111
Markdown                         1              5              0             40
DOS Batch                        2              1              0              9
-------------------------------------------------------------------------------
SUM:                           123           2882            390          15312
-------------------------------------------------------------------------------
```

Figure 24 - Analysis of the RCP after deployment (CLOC)

Table 13 - RiverCure Portal - lines of code statistics

RiverCure Portal				
ASL Specifications		Application		
Manual	Generated	Manual	Generated (CLOC)	Deployed Files (CLOC)
390	2090	14289	1023	15312

The code generation ratio through the ASL Approach is **1023** (Application Generated (CLOC)) / **390** (Initial ASL Specifications) = **2,62** (more than the double).

After all the manual code changes for the development and the deployment in a Windows Server, the MyTinyBlog has a total of **15312** lines of code, so the coverage of the ASL Approach is **1023/15312** = **6%**.

The difference of the values between the MyTinyBlog (**81%**) and RiverCure Portal (**6%**) was expected since the development of RiverCure Portal was mostly achieved through traditional coding practices except for some parts as we have explained in this section.

6.3. Comparison with the Related Work

Mendix and Outsystems platforms surpass ASL transformations by providing a user-friendly interface that allows development of applications with features and customization aspects. ASL provides a good start for many situations due to its flexibility and extensibility. Using a lower code-level, it can be challenging for people that do not usually work with programming languages and other IT tools. Still, it may simplify the communication of the software application's vision. From the generated application, we still have control over the necessary code to scale the web application.

Like ASL, EMF on Rails it accelerates the generation of CRUD operations on data models [45]. A difference of our project and EMF on Rails transformations is when their impact is more visible, as ASL promotes a better understanding of requirements at the start and final specifications through interfaces and use-cases specifications. ADM (Ariadne Development Method) is another approach with the primary goal of accelerating the development of web systems [68]. Like ASL, ADM offers constructs to specify these systems making use of Labyrinth++. This tool allows the specification of all the components for web systems and includes a pattern language. Those patterns are organized according to the nature of the problem they solve and make the development of solution easier for less-experienced web developers [68].

Like RSL, but on the contrary of IFML, the concrete syntax of ASL specs are textual and consequently more natural to be rigorously defined and validated. ASL adapts the XIS smart approach, where UI models can be generated from high-level models.

Unlike XIS, ASL can allow to specify and to automatize the process of creating different types of users, assigning distinct roles and respective permissions. Due to its platform-independent and human-friendly text-based syntax, ASL specifications are more open and easier to be manipulated and interoperated comparing with the options referred above, namely the commercial solutions. One relevant work to explore in the future is to verify if ASL could be suitable to support interoperability between the models developed with these low-code or no-code platforms.

7. Conclusion

This dissertation proposes an approach that intends to address the followings issues: How to combine the specification of requirements and the design of business applications' in an integrated way and how to increase the productivity of developers by automatizing the production of artifacts like technical documentation and software code. The ASL-based approach combines the disciplines of requirements engineering and model-driven engineering. This approach is based on existing solutions, namely those mostly related to the RSL and IFML languages, in which the ASL design is based. ASL allows to rigorously specify requirements (namely use cases with their relationships with actors and data entities), but also to specify user interfaces of applications. We show that this language can be properly supported by tools like the ITLingo Studio, also model-to-model and model-to-code transformations, and thus can considerably improve the quality and productivity of both the requirements definition and the development of these applications. We support our proposal and discussion with two case studies, developed on top of a popular Python based framework, the Django framework.

7.1. Contributions

To achieve the objectives of this research, the most important contributions of this dissertation can be summarized as follows: (i) updates and tests of the ASL Language, developed in Xtext framework by Professor Alberto Rodrigues da Silva, the original creator of the language; (ii) development of a cohesive set of Xtend files that allows to implement a first generation of the M2M and M2C transformations, respectively from ASL specifications and into new ASL specifications and into Django artifacts; (iii) along with Jorge Marques, design and development of the RiverCure Portal (RCP), which main goal is to improve water resources management and protection. I focused in the development of the models, CRUD operations, groups and permissions for each module using ASL M2C transformations; (iv) publication of a scientific article in QUATIC 2020 and its presentation at 10th September [72].

7.2. Future Work

Future research shall consider improving the customization of either the specification and generation of the business applications and shall specify and develop more cases studies. The integration or combination with other popular software (e.g., NodeJS, JavaScript frameworks, .NET) and low-code frameworks (e.g., Mendix, OutSystems) can also be considered as they may bring more flexibility to this solution.

As seen in Chapter 6, we were able to generate many code files that contain namely ASL specifications and Python code, but there were repeated blocks of specifications that ended up not being viable with the complex features of case study B, such as generated ASL UI specifications. This shows that the ASL M2M transformation needs to be optimized. These improvements can be dealt

with by analyzing the ASL model-to-model process and finding new patterns of model-to-code transformations for other languages and frameworks.

Since the current version of the IT-Lingo-Studio is not a friendly tool to unexperienced users or people who have no IT background, a more interactive development could be considered namely to generate parts of web applications, similarly to some of the related work approaches.

Appendix A

Generated ASL UIs for MyTinyBlog Use Case uc_2_ManagePosts "Manage Blog Posts"

```
Package p_MyTinyBlog

Import p_MyTinyBlog.MyTinyBlog
Import p_MyTinyBlog.MyTinyBlog.*

System UI_MyTinyBlog : Application [ ]

//UIContainer aligned with uc_2_ManagePosts
//MAIN WINDOW UI

UIContainer uiCt_e_Post : Window [

   component uiCo_e_PostList : List: List_Table [
     isScrollable
     dataBinding e_Post [
       visualizationAttributes  e_Post.author, e_Post.category, e_Post.title,
       sortAttributes  e_Post.author, e_Post.category, e_Post.title,
       //orderBy e_Post.author e_Post.author e_Post.author  DESC
     ]

   part uip_checkselect: Field: Field_Input : WFC_CheckBox
]

component uiCo_Filter_e_Post: Details [
     dataBinding e_Post [
       filterAttributes , e_Post.author, e_Post.category, e_Post.title
     ]
   ]

   component uiCo_Search_e_Post: Details [
     dataBinding e_Post [
       searchAttributes , e_Post.author, e_Post.category, e_Post.title
     ]
   ]

component uiCo_Actions: Menu [

     event uiev_create "Create e_Post" : Submit: Submit_Create [navigationFlowTo
uiCt_e_PostCreator]
     event uiev_delete "Delete e_Post" : Submit: Submit_Delete
     event uiev_update "Edit" : Submit: Submit_Update [navigationFlowTo uiCt_e_PostCreator]
   ]

]
//UC Type: EntitiesManage

//UI FOR SEARCH

component uiCo_Search: Details [
   part uip_search: Field: Field_Output: WFC_Label [Text defaultValue "Search:"]
   part uip_search_e_Post: Field: Field_Input : WFC_Text [Text defaultValue "Search  e_Post"]
   event uiev_search "Search": Submit: Submit_Ok
]
// UI FOR DELETE

UIContainer uiCt_Delete_Warning : Window  [
   component uiCo_DeleteWarning : Dialog : Dialog_Warning  [

     part uip_Delete_Warning_Text: Field: Field_Output [Text defaultValue "Are you sure you
want to delete e_Post?"]
     event uiev_delete : Submit: Submit_Delete [navigationFlowTo uiCt_e_Post]
     event uiev_cancel : Submit: Submit_Cancel [navigationFlowTo uiCt_e_Post]
     ]
   ]

UIContainer uiCt_Delete_Confirmation : Window : Window_Modal [
   component uiCo_Delete_Message : Dialog : Dialog_Info [
      part uip_Delete_Confirmation_Text: Field: Field_Output [Text defaultValue "e_Post
Deleted"]
```

```
      event uiev_confirm: Submit : Submit_Ok [navigationFlowTo uiCt_e_Post]
     ]
]

UIContainer uiCt_e_PostReader : Window: Window_Modal[

   component uiCo_Edite_Post : Form : Form_Simple [
   part uip_authorLabel  : Field: Field_Output : WFC_Label [Text defaultValue "author"]
   part uip_authorValue : Field: Field_Input : WFC_Text[Text defaultValue "Editable Value"]

   part uip_categoryLabel  : Field: Field_Output : WFC_Label [Text defaultValue "category"]
   part uip_categoryValue : Field: Field_Input : WFC_Text[Text defaultValue "Editable Value"]

   part uip_dateLabel  : Field: Field_Output : WFC_Label [Text defaultValue "date"]
   part uip_dateValue : Field: Field_Input : WFC_Text[Text defaultValue "Editable Value"]
   part uip_BodyLabel  : Field: Field_Output : WFC_Label [Text defaultValue "Body"]
   part uip_BodyValue : Field: Field_Input : WFC_Text[Text defaultValue "Editable Value"]
   part uip_titleLabel  : Field: Field_Output : WFC_Label [Text defaultValue "title"]
   part uip_titleValue : Field: Field_Input : WFC_Text[Text defaultValue "Editable Value"]
   part uip_stateLabel  : Field: Field_Output : WFC_Label [Text defaultValue "state"]
   part uip_stateValue : Field: Field_Input : WFC_Text[Text defaultValue "Editable Value"]
   ]
component uiCo_e_PostNavigationButtons: Details [

   event uiev_beginning_e_Post "Beginning": Submit: Other [navigationFlowTo uiCt_e_PostReader]
   event uiev_previous_e_Post "Previous":   Submit: Other [navigationFlowTo uiCt_e_PostReader]
   event uiev_next_e_Post "Next": Submit: Other [navigationFlowTo uiCt_e_PostReader]
   event uiev_end_e_Post "End": Submit: Other [navigationFlowTo uiCt_e_PostReader]
   event uiev_back "Back#: Submit: Other [navigationFlowTo uiCt_e_Post]
]

]
//UI FOR CREATE/UPDATE

   UIContainer uiCt_e_PostCreator : Window : Window_Modal[
   component uiCo_Edite_Post : Form : Form_Simple [
        part uip_authorLabel : Field: Field_Output : WFC_Label [Text defaultValue "author"]
        part uip_authorValue : Field: Field_Input : WFC_Text[Text defaultValue "Editable
Value"]
        part uip_categoryLabel : Field: Field_Output : WFC_Label [Text defaultValue "category"]
        part uip_categoryValue : Field: Field_Input : WFC_Text[Text defaultValue "Editable
Value"]
        part uip_dateLabel : Field: Field_Output : WFC_Label [Text defaultValue "date"]
        part uip_dateValue : Field: Field_Input : WFC_Text[Text defaultValue "Editable Value"]

        part uip_BodyLabel : Field: Field_Output : WFC_Label [Text defaultValue "Body"]
        part uip_BodyValue : Field: Field_Input : WFC_Text[Text defaultValue "Editable Value"]

        part uip_titleLabel : Field: Field_Output : WFC_Label [Text defaultValue "title"]
        part uip_titleValue : Field: Field_Input : WFC_Text[Text defaultValue "Editable Value"]

        part uip_stateLabel : Field: Field_Output : WFC_Label [Text defaultValue "state"]
        part uip_stateValue : Field: Field_Input : WFC_Text[Text defaultValue "Editable Value"]

        event uiev_create "Create e_Post" : Submit : Submit_Create [navigationFlowTo
uiCt_e_Post]
        event uiev_cancel "Cancel e_Post" : Submit : Submit_Cancel [navigationFlowTo
uiCt_e_Post]
        ]

        ]
```

Appendix B

RiverCure Portal ASL Specification (complete):

```
 Package RiverCure
/*******************************************************
     System definition
*******************************************************/

System RiverCurePortal "RiverCure Portal" : Application: Application_Web [isFinal
description
"RiverCurePortal shall provide data from different sources, such as SNIRH (APA's system),
photos obtained
from social media and from the simulation tool, HiStav"]

/*******************************************************
   DataAttributeType
*******************************************************/
DataAttributeType GeoPoint  [description "GeoPoint attribute type (for geospatial data)"]
DataAttributeType GeoPolyline [description "GeoPolyline attribute type (for geospatial data)"]
DataAttributeType GeoPolygon [description "GeoPolygon attribute type (for geospatial data)"]
DataAttributeType GeoRaster [description "GeoRaster attribute type (for geospatial data)"]

/*******************************************************
   DataEnumerations
*******************************************************/
DataEnumeration SensorKind "Sensor Kind" values ( "HydrometricSensor", "WeatherSensor",
"SocialNetworkScanner", "HumanSensor", "TBD Sensor" )
DataEnumeration SensorModalityKind "Sensor Modality Kind" values ( "PhysicalFixed ",
"PhysicalMobile", "DigitalSocialNetworkScanner", "DigitalHumanUpload" )
DataEnumeration TimeserieType "Timeserie type"  values ("Continuous", "Discontinuous",
"Statistical")
DataEnumeration HydroFeatureKind "HydrometricFeature Kind" values ("River", "Estuary", "Lake",
"RiverBasin", "DrainageBasin", "Dam")
DataEnumeration GeometryKind "Geometry Kind" values ("Point", "Polyline", "Polygon")
DataEnumeration SimulationKind "Simulation Kind" values ("Simulation", "Scenario")
DataEnumeration InformationKind "Information Kind" values ("Simulation", "Event", "Sensor",
"Alarm")
DataEnumeration CityKind values ("City", "Town", "Village", "Other")
DataEnumeration OrganizationKind values ("WaterAuthority", "Municipality", "ResearchLab",
"Partner", "Other")
DataEnumeration AlarmPermission "Alarm Permission" values ("Yes", "Yes (only authorities
alarms)", "No", "Depend on secondary role")
DataEnumeration MetricKind "Metric" values ("Sec", "Min", "Hour", "Day", "Week", "Month",
"Year")
DataEnumeration ColourKind "Colour" values ("Red", "Yellow", "Green")
DataEnumeration ContextBoundaryLineKind values ("Input", "Output", "InputOutput")
DataEnumeration ContextBoundaryLineDataKind values ("Depth (H)", "Discharge (Q)", "Velocity
(V)", "Elevation (Z)")
DataEnumeration EventKind values ("Flood", "HeavyPrecipitation", "HydrologicalDrought",
"MeteorologicalDrought", "Hurricane", "Tsunami", "Storm", "LandSlide")
DataEnumeration EventState  values ("Announced", "Occurring", "Concluded")
DataEnumeration EventSubKind values ("Forecast", "Hindcast", "Planning")
DataEnumeration AccessRequestState values ("Processing", "Finished")
DataEnumeration ContextUserKind values ("ContextAdmin","ContextManager", "Visitor")

/*******************************************************
   DataEntities
*******************************************************/

/*******************************************************
 General geographic entities, e.g. Country, District, ...
***/
DataEntity e_Country "Country" : Reference [
  attribute id "Id" : Integer [constraints (PrimaryKey)]
  attribute code "Code ISO-2" : Regex [ constraints( NotNull Unique Check
(RegexValidationExpression "r'^[A-Z][A-Z]")) ]
  attribute Name "Name" : String [constraints(NotNull Unique)]
  attribute capital "Capital" : Integer [constraints(NotNull Unique ForeignKey(e_City))]
  attribute geom "Geometry" : GeoPolygon
  constraints (showAs (Name))
]
```

```
DataEntity e_District "District" : Reference [
  attribute id "Id" : Integer [constraints (PrimaryKey)]
  attribute code "Di Code" : Regex [ constraints( NotNull Unique Check
(RegexValidationExpression "r'^PT[0-9][0-9]")) ]
  attribute Name "Name" : String [constraints(NotNull Unique)]
  attribute capital "Capital" : Integer [constraints(NotNull Unique ForeignKey(e_City))]
  attribute country "Country" : Integer [constraints(NotNull ForeignKey(e_Country))]
  attribute geom "Geometry" : GeoPolygon
  constraints (showAs (Name))
]

DataEntity e_Municipality "Municipality" : Reference [
  attribute id "Id" : Integer [constraints (PrimaryKey)]
  attribute code "Dico Code" : Regex [constraints(NotNull Unique)]
  attribute Name "Name" : String [constraints(NotNull Unique)]
  attribute district "District" : Integer [constraints(NotNull ForeignKey(e_District))]
  attribute capital "Capital" : Integer [constraints(NotNull Unique ForeignKey(e_City))]
  attribute geom "Geometry" : GeoPolygon
  constraints (showAs (Name))
]

DataEntity e_Parish "Parish" : Reference [
  attribute id "Id" : Integer [constraints (PrimaryKey)]
  attribute code "Dicofre Code" : Regex [constraints(NotNull Unique)]
  attribute Name "Parish name" : String [constraints(NotNull)]
  attribute municipality "Municipality" : Integer [constraints(NotNull ForeignKey
(e_Municipality))]
  attribute geom "Geometry" : GeoPolygon
  constraints (showAs (Name))
]

DataEntity e_City "City" : Reference [
  attribute id "Id" : Integer [constraints (PrimaryKey)]
  attribute code "Code" : Regex [constraints(NotNull)]
  attribute Name "Name" : String [constraints(NotNull)]
  attribute type "Type" : DataEnumeration CityKind [constraints(NotNull)]
  attribute municipality "Municipality" : Integer [constraints(NotNull ForeignKey
(e_Municipality))]
  attribute geom "Geometry" : GeoPoint
  constraints (showAs (Name))
]

//HydroFeature: River, Lake, RiverBasin, ...
DataEntity e_HydroFeature "Hydrometric feature" : Master [
  attribute id "Id" : Integer [constraints (PrimaryKey)]
  attribute code "HydroFeature Code" : Regex [constraints(NotNull Unique)]
  attribute Name "Name" : String [constraints(NotNull )]
  attribute type "Type" : DataEnumeration HydroFeatureKind [constraints(NotNull)]
//attribute shape "Shape" : DataEnumeration GeometryKind [constraints(NotNull)]
//attribute dico "Municipality code" : Regex [constraints(NotNull Unique ForeignKey(
e_Municipality))]

  // just for e_RiverBasin, Dam, ...
  attribute area "Area" : Decimal
  //attribute alarmName "Alarm name" : String [constraints(ForeignKey (e_Alarm))]
  // just for e_River, ...
  attribute length "Length" : Decimal
  //attribute hierarchy "Hierarchy" : String

  attribute PartOf "PartOf" : Integer [constraints(ForeignKey (e_HydroFeature))]
  attribute flowsInto "Flows Into" : Integer [constraints(ForeignKey (e_HydroFeature))]

  attribute geom "Geometry" : GeoPolygon
  constraints (showAs (Name))
]

/*********************************************************
 Organization entity: Organization
***/
DataEntity e_Organization "Entity" : Master [
  attribute id "Id" : Integer [constraints (PrimaryKey)]
  attribute Name "Name" : String [constraints(NotNull Unique)]
  attribute type "Type" : DataEnumeration OrganizationKind [constraints(NotNull)]
  attribute sector "Sector" : String [constraints(NotNull)]

  attribute address "Address" : String [constraints(multiplicity "0..2" Encrypted)]
  attribute city "City" : Integer [constraints(ForeignKey(e_City))]
```

```
   attribute country "Country" : Integer [constraints(NotNull Encrypted ForeignKey(e_Country))]
   attribute email "Email" : Email [constraints(multiplicity "0..2" Encrypted)]
   attribute phone "Telephone" : String [constraints(multiplicity "0..2" Encrypted)]

   attribute geom "Geometry" : GeoPoint [constraints (NotNull)]
   constraints (showAs (Name))
]

/*************************************************************
 Users: General Sensor, e_FixedInSituSensor, e_WeatherSensor, e_PhotoSensor
***/
DataEntity e_User "User" : Master [
  //attribute id "ID" : Integer [constraints (PrimaryKey)]
  attribute login "Login" : String [constraints (NotNull Unique)]
  attribute password "Password" : Regex [constraints (NotNull Encrypted Check
(RegexValidationExpression "r'[A-Za-z0-9@#$]{6,12}'"))]
  attribute Name "Name" : Text [constraints (NotNull Encrypted)]
  attribute email "Main Email" : Email [constraints (Unique Encrypted)]
  //attribute state "State" : DataEnumeration UserState [constraints(NotNull)]
  // attribute userProfileID "User Profile" : Integer [constraints (NotNull ForeignKey
(e_UserProfile onDelete PROTECT))]

  attribute emails "Email address(es)" : Email [constraints(multiplicity "0..3" NotNull Unique
Encrypted )]
  attribute phoneNumbers "Telephone number(s)" : Regex [constraints(multiplicity "0..3"
NotNull Unique Encrypted Check (RegexValidationExpression "r'^([0-9]{9}")) ]
  constraints (showAs (login))
  tag (name "tenant" value "user")
]

/*************************************************************
 Sensors: General Sensor, e_FixedInSituSensor, e_WeatherSensor, e_PhotoSensor
***/
DataEntity e_Sensor "Sensor" : Master [
   attribute id "Id" : Integer [constraints (PrimaryKey)]
   attribute code "code" : String [constraints (NotNull Unique)]
   attribute Name "name" : String [constraints (NotNull)]
   attribute type "Type" : DataEnumeration SensorKind [constraints (NotNull)]
   attribute modalityType "Modality Type" : DataEnumeration SensorModalityKind [constraints
(NotNull)]
   attribute Description "Description" : Text
   attribute Version "Version" : String
   attribute responsible "User Responsible" : Integer [constraints(NotNull ForeignKey(e_User))]
   attribute geom "Geometry" : GeoPoint


// FixedInSituSensor
   attribute recRhythmValue "Recording rhythm value" : Integer []
   attribute recRhythmMetric "Recording rhythm metric" : DataEnumeration MetricKind

// HydrometricSensor
   attribute zeroLevelScale "Zero level scale" : Decimal

// WeatherSensor
   attribute maxRange "Max range" : Integer
   attribute PRF "Pulse repetition frequency (PRF)" : Integer
   attribute recRhythm "Recording rhythm" : String

// PhotoSensor
   attribute isSocialNetwork "is from a social network" : Boolean [defaultValue "False"]
   attribute isUpload "is from an upload" : Boolean [defaultValue "False"]
   attribute source "Source" : String
   attribute url "URL" : URL
   attribute isAgreeTerms "is agreed with terms and conditions" : Boolean [defaultValue "True"]
]


// SensorAlarm
DataEntity e_SensorAlarm "Sensor Alarm" : Master [
   attribute id "Id" : Integer [constraints (PrimaryKey)]
   attribute sensorId "Sensor" : Integer [constraints (NotNull ForeignKey (e_Sensor))]
   attribute Name "Name" : String [constraints (NotNull)]
   attribute Action "Action" : String
   attribute Description "Description" : Text
   attribute colour "Colour" : DataEnumeration ColourKind [ constraints ( NotNull Derived
("TBD"))]
```

```
  attribute redMinthreshold "Low threshold" : Decimal [constraints (NotNull)]
  attribute redMaxthreshold "Up threshold" : Decimal [constraints (NotNull)]
  attribute yellowMinthreshold "Low threshold" : Decimal [constraints (NotNull)]
  attribute yellowMaxthreshold "Up threshold" : Decimal [constraints (NotNull)]
  attribute greenMinthreshold "Low threshold" : Decimal [constraints (NotNull)]
  attribute greenMaxthreshold "Up threshold" : Decimal [constraints (NotNull)]
]

DataEntity e_SensorObservation "Sensor observation" : Document [
  attribute id "Id" : Integer [constraints (PrimaryKey)]
  attribute sensorId "Sensor" : Integer [constraints (NotNull ForeignKey (e_Sensor))]
  //attribute sensorType "Sensor Type" : DataEnumeration SensorKind [constraints (Derived
("sensorId.type"))]

//attribute startDatetime "Start Datetime" : Datetime [constraints (NotNull)]
//attribute endDatetime "End Datetime" : Datetime [constraints (NotNull)]
  attribute date "Date" : Date[constraints (NotNull)]

  attribute time "time" : Time [constraints (NotNull)]

  // TBD

  attribute data "Data" : String  //dados

  // HydrometricSensor
  attribute depth : Double       // profundidade (m)
  attribute discharge : Double // caudal (m3/seg)
  attribute volume : Double    // volume (m3)
  attribute velocity : Double    // velocidade (m/seg)
  attribute elevation : Double // cota (m)

  // WeatherSensorObservation
  attribute rainfall : Double         // precipitação (m)
  attribute soilWaterContent : Double   // teor em água do solo (%)

  //HumanSensorObservation
  attribute photo : Image
  attribute geom "Geometry" : GeoPoint [constraints (NotNull)]
  attribute elevation : Double [constraints (Derived ("ML techniques from Photo"))]       //
cota (m3)
  attribute velocity : Double  [constraints (Derived ("ML techniques from Photo"))]       //
velocidade (m/seg)

  //RadarSensorObservation
  // Mapa de valores
  attribute rainfall : Double         // precipitação (m)

  //TBD SensorObservation
  attribute isCummulative "is Cummulative" : Boolean [defaultValue "False"]
  attribute nValueTotal "Total number of values" : Integer [constraints (NotNull)]

  //e_PhotoSensorObservation
  attribute url "URL" : URL
  attribute fileName "File name" : String
  attribute height "Height" : Integer
  attribute horizontalRes "Horizontal resolution" : Integer
  attribute verticalRes "Vertical resolution" : Integer
  attribute nBits "Number of bits" : Integer
  attribute width "Width" : Integer
  attribute fileFormat "File format" : String
]

/*********************************************************
 Core entities: Context, ContextBoundaryLine, ... ContextOrganization,
ContextOrganizationUser, ...
***/
DataEntity e_Context "Context" : Master [
  attribute id "Id" : Integer [constraints (PrimaryKey)]
  attribute code "Code" : String [constraints(NotNull Unique)]
  attribute Name "Name" : String [constraints(NotNull Unique)]
  // attribute country "Country" : Integer [constraints(NotNull ForeignKey(e_Country))]
  attribute hydroFeature "HydroFeature" : Integer [constraints(NotNull
ForeignKey(e_HydroFeature))]

  //attribute geomExternalBoundary : Integer  [constraints(NotNull ForeignKey(e_Geometry))]
```

```
  //the context shall have internal and external boundaries, an alignment line, and several
boundary-lines
  attribute geomExternalBoundary "ExternalBoundary Geometry" : GeoPolygon
         // aka Domain
  attribute CLExternalBoundary "ExternalBoundary CL" : Double
       // aka Domain's CL, characteristic lenght

  attribute geomInternalBoundary "InternalBoundary Geometry" : GeoPolygon
[constraints(multiplicity"1..*")] // aka Refinement
  attribute CLInternalBoundary "InternalBoundary CL" : Double
[constraints(multiplicity"1..*")]  // aka Refinement's CL

  attribute geomAlignment "Alignment Geometry" : GeoPolyline
[constraints(multiplicity"1..*")]// aka Alignment
  attribute CLAlignment "Alignment CL" : Double   [constraints(multiplicity"1..*")]
       // Alignment's CL

  attribute userResponsible "User Responsible" : Integer [constraints(NotNull
ForeignKey(e_User))]

  attribute isPublic "Context Access Restrictions" : Boolean [ defaultValue "False"
constraints (NotNull )]

  tag (name "tenant-main" value "true")

]

DataEntity e_ContextBoundaryLine : Master [
  attribute id "Id" : Integer [constraints (PrimaryKey)]
  attribute context "Context" : Integer [constraints(NotNull ForeignKey(e_Context))]
  attribute geom : GeoPolyline [constraints (NotNull Check (superimposed "must be a line
superimposed on context.geomExternalBoundary"))]
  attribute type : DataEnumeration ContextBoundaryLineKind [constraints (NotNull)]
  attribute dataType: DataEnumeration ContextBoundaryLineDataKind
]

DataEntity e_ContextBoundaryPoint : Master [
  attribute id "Id" : Integer [constraints (PrimaryKey)]
  attribute contextBoundaryLine "Context BoundaryLine" : Integer [constraints(NotNull
ForeignKey(e_ContextBoundaryLine))]
  attribute geom : GeoPoint [constraints (NotNull Check (superimposed "must be a point
superimposed on e_ContextBoundaryLine.geom"))]
  //attribute Sensor "Sensor" : Integer [constraints(NotNull ForeignKey(e_ContextSensor))]
]

DataEntity e_ContextBoundaryPointSensor : Master [
  attribute id "Id" : Integer [constraints (PrimaryKey)]
  attribute point "Context BoundaryPoint" : Integer [constraints(NotNull
ForeignKey(e_ContextBoundaryPoint))]
  attribute Sensor "Sensor" : Integer [constraints(NotNull ForeignKey(e_ContextSensor))]
]

/*************************************************************
 ContextSensor entity: ContextSensor
***/
DataEntity e_ContextSensor "ContextSensor" : Master [
  attribute id "Id" : Integer [constraints (PrimaryKey)]
  attribute context "Context" : Integer [constraints(NotNull ForeignKey(e_Context))]
  attribute Sensor "Sensor" : Integer [constraints(NotNull ForeignKey(e_Sensor))]
  attribute Description "Description" : Text

  attribute associateDatetime "Associate Datetime" : Datetime [constraints(NotNull)]
  attribute associateUser "Associate User" : String [constraints(NotNull ForeignKey (e_User))]
]

/*************************************************************
 Context's Events
***/

DataEntity e_ContextEvent "Event" : Master [
  attribute id "Id" : Integer [constraints (PrimaryKey)]
  attribute context "Context" : Integer [constraints(NotNull ForeignKey(e_Context))]
  attribute Name "Name" : String [constraints (PrimaryKey)]
  attribute type "Event Type" : DataEnumeration EventKind [constraints(NotNull)]
  attribute subType "Event SubType" : DataEnumeration EventSubKind [constraints(NotNull)]
  attribute state "State" : DataEnumeration EventState [constraints(NotNull)]
```

```
  attribute startDatetime "Start Event" : Datetime [constraints(NotNull)]
  attribute endDatetime "End Event" : Datetime [constraints(Check (ck_datesValidation
"endDatetime >= startDatetime"))]
  attribute Description "Description" : Text

  // Attributes for "Flood Simulation" event, with HiStav
  attribute returnPeriod "Return Period" : Integer // n° of years ...
  attribute warmUp "Warm up" : Boolean
  //attribute simulationType "Simulation Type" : DataEnumeration EventKind
[constraints(NotNull)]

]
/************************************************************
 Context's Organizations and Users
***/

//Associated to ContestAccessRequest
DataEntity e_ContextUser "ContextUser" : Parameter [
  attribute id "Id" : Integer [constraints (PrimaryKey)]
  attribute context "Context" : Integer [constraints(NotNull ForeignKey(e_Context))]
  attribute user "User" : Integer [constraints(NotNull ForeignKey(e_User))]
  attribute contextUserRole "User Role" : DataEnumeration ContextUserKind [constraints
(NotNull)]

  tag (name "tenant-detail" value "true")
]


DataEntity e_ContextAccessRequest "Context Request Access" : Parameter [
  attribute context "Context" : Integer [constraints(NotNull ForeignKey(e_Context))]
  attribute access_granted "Access granted": Boolean [defaultValue "False"]
  attribute state "request state" : DataEnumeration AccessRequestState [constraints (NotNull)]
  attribute requestUser "request user" : String [constraints(NotNull ForeignKey(e_User))]
  attribute type "access type" : DataEnumeration ContextUserKind [constraints (NotNull)]
]


/*******************************************************
   DataEntityClusters
*******************************************************/

DataEntityCluster ec_Country "Country" : Reference [main e_Country uses e_City]
DataEntityCluster ec_District "District" : Reference [main e_District child e_Municipality
[uses e_City] uses e_City]
DataEntityCluster ec_Municipality "Municipality" : Reference [main e_Municipality child
e_Parish uses e_City]
DataEntityCluster ec_City "City" : Reference [main e_City]
DataEntityCluster ec_Organization "Organization" : Master [main e_Organization uses e_Country]
DataEntityCluster ec_User "User" : Master [main e_User ]//child e_UserSettings]
DataEntityCluster ec_Sensor "Sensor Simple" : Master [main e_Sensor]
DataEntityCluster ec_SensorComplete "Sensor" : Master [main e_Sensor  child
e_SensorObservation] //child e_SensorAlarm]
DataEntityCluster ec_HydroFeature "HydroFeature" : Master [main e_HydroFeature]

DataEntityCluster ec_ContextEvent "Event" : Master [main e_ContextEvent child e_SensorAlarm ]

/*******************************************************
   Actors view
*******************************************************/

ContextActor aU_Admin "Administrator" : User [description "manages Users and organizations,
configures settings"]
ContextActor aU_Manager "Manager" : User [description "manages geographic entities (e.g.,
country, city, ..., hydro-feature; creates context"]
ContextActor aU_SensorsManager "SensorsManager" : User [description "manages sensors"]
ContextActor aU_User "User" : User [description "consult general info, see maps with context's
boundary, hydro-features, ..."]
ContextActor aU_ContextAdmin "Context Administrator" : User [description "manage admin
context, namely associate organizations and users"]
ContextActor aU_ContextManager "Context Manager" : User [description "define geo-elements of
the context, associate sensors"]
ContextActor aU_ContextEventManager "Context Event Manager" : User [description "manage
events, simulations"]

UseCase uc_1_ManageUsers "Manage Users" : EntitiesManage [
  actorInitiates aU_Admin
  dataEntity ec_User //Users, Groups, Permissions
```

```
  actions aCreate, aRead, aUpdate, aDelete

]

UseCase uc_2_ManageOrganizations "Manage Organizations" : EntitiesManage [
  actorInitiates aU_Admin
  dataEntity ec_Organization
  actions aCreate, aRead, aUpdate, aDelete
]

UseCase uc_3_ManageLocations "Manage Locations" : EntitiesManage [
  actorInitiates aU_Manager
]

UseCase uc_3_1_ManageCountries "Manage Countries" : EntitiesManage [
  actorInitiates aU_Manager
  dataEntity ec_Country
  actions aCreate, aRead, aUpdate, aDelete
]

UseCase uc_3_2_ManageCities "Manage Cities" : EntitiesManage [
  actorInitiates aU_Manager
  dataEntity ec_City
  actions aCreate, aRead, aUpdate, aDelete
]

UseCase uc_3_3_ManageMunicipalities "Manage Municipalities" : EntitiesManage [
  actorInitiates aU_Manager
  dataEntity ec_Municipality
  actions aCreate, aRead, aUpdate, aDelete
]

UseCase uc_3_4_ManageParishes "Manage Parishes" : EntitiesManage [
  actorInitiates aU_Manager
  dataEntity e_Parish
  actions aCreate, aRead, aUpdate, aDelete
]

UseCase uc_5_ManageHydrofeatures "Manage HydroFeatures" : EntitiesManage [
  actorInitiates aU_Manager
  dataEntity e_HydroFeature
  actions aCreate, aRead, aUpdate, aDelete
]

UseCase uc_7_ManageSensors "Manage sensors" : EntitiesBrowse [
  actorInitiates aU_SensorsManager
  dataEntity ec_Sensor
  actions aRead
]

UseCase uc_7_1_ManageSensor "Manage a single sensor" : EntitiesManage [
  actorInitiates aU_SensorsManager
  dataEntity ec_SensorComplete
  actions aCreate, aRead, aUpdate, aDelete
  extensionPoints xp_ManageSensorData, xpManageSensorAlarms
]

UseCase uc_7_1_1_ManageSensorData "Manage Sensor Observations" : EntitiesManage [
  actorInitiates aU_SensorsManager
  dataEntity e_SensorObservation
  actions aCreate, aRead, aUpdate, aDelete
  extends uc_7_1_ManageSensor onExtensionPoint uc_7_1_ManageSensor.xp_ManageSensorData
]

UseCase uc_7_1_2_ManageSensorAlarms "Manage Sensor Alarms" : EntitiesManage [
  actorInitiates aU_SensorsManager
  dataEntity e_SensorAlarm
  actions aCreate, aRead, aUpdate, aDelete
  extends uc_7_1_ManageSensor onExtensionPoint uc_7_1_ManageSensor.xpManageSensorAlarms
]

UseCase uc_6_ManageContexts "Manage contexts" : EntitiesBrowse [
  actorInitiates aU_ContextAdmin
  dataEntity e_Context
  actions aRead
]
```

```
UseCase uc_6_1_AdminContext "Manage a single context, namely updating info, deleting and
associating organizations/users" : EntityUpdate [
   actorInitiates aU_ContextAdmin
   dataEntity e_Context
   actions aDelete, aRead, aUpdate
]

UseCase uc_6_2_ManageContext "define geo-elements of the context, associate sensors" :
EntityUpdate [
   actorInitiates aU_ContextManager
   dataEntity e_Context
   actions aRead, aUpdate
   extensionPoints xp_ManageEvents
]

UseCase uc_6_2_1_ManageEvents "Manage Events" : EntitiesManage [
   actorInitiates aU_ContextEventManager
   dataEntity ec_ContextEvent
   actions aCreate, aRead, aUpdate, aDelete
   extends uc_6_2_ManageContext onExtensionPoint uc_6_2_ManageContext.xp_ManageEvents
]

UseCase uc_13_ConsultGeneralInfo "Consult general info" : EntitiesBrowse [
   actorInitiates aU_User
   extensionPoints xp_ConsultHydrofeatures, xp_ConsultSensorData
]

UseCase uc_13_1_ConsultHydrofeatures "Consult hydrofeatures" : EntitiesBrowse [
   actorInitiates aU_User
   dataEntity e_HydroFeature
   actions aRead
   extends uc_13_ConsultGeneralInfo onExtensionPoint
uc_13_ConsultGeneralInfo.xp_ConsultHydrofeatures
]

UseCase uc_13_2_ConsultSensorData "Consult sensor data" : EntitiesBrowse [
   actorInitiates aU_User
   dataEntity e_SensorObservation
   actions aRead
   extends uc_13_ConsultGeneralInfo onExtensionPoint
uc_13_ConsultGeneralInfo.xp_ConsultSensorData
]
```

Appendix C

RiverCure Portal Screenshots:

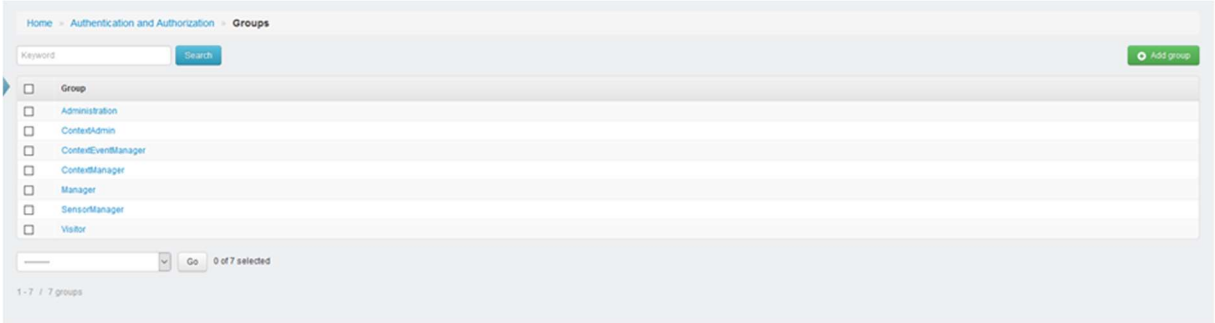

Figure 25 - Administration page in Django Admin

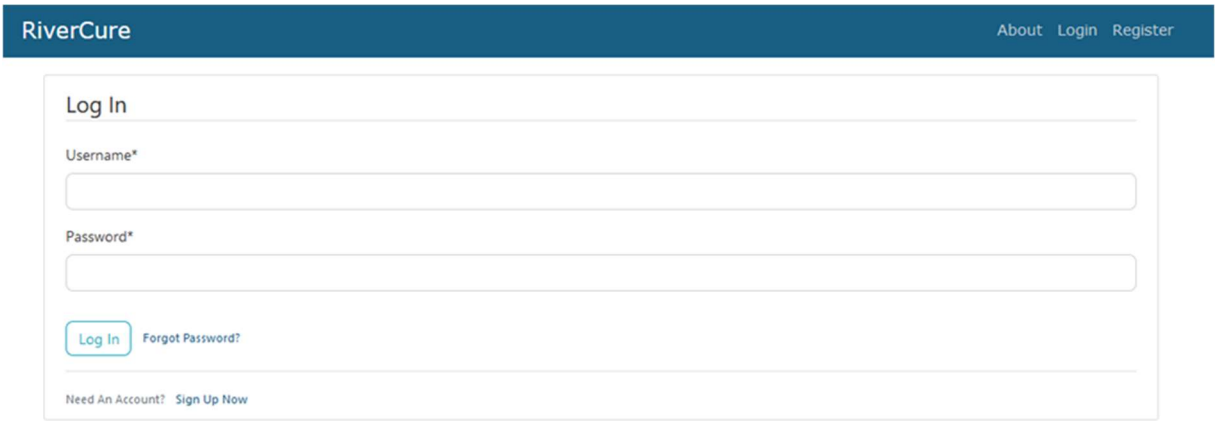

Figure 26 - RiverCure Portal (Log In Page)

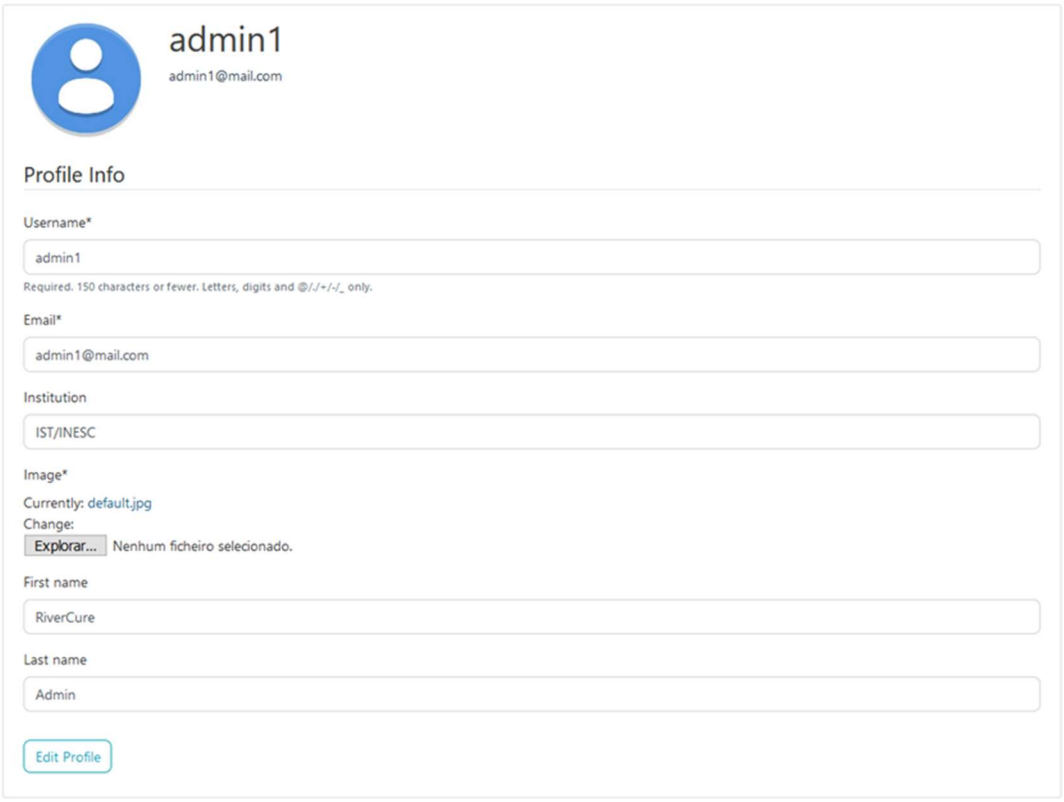

Figure 27-RCP User Profile Page

Figure 28- Hydro Feature Detail Page in RCP

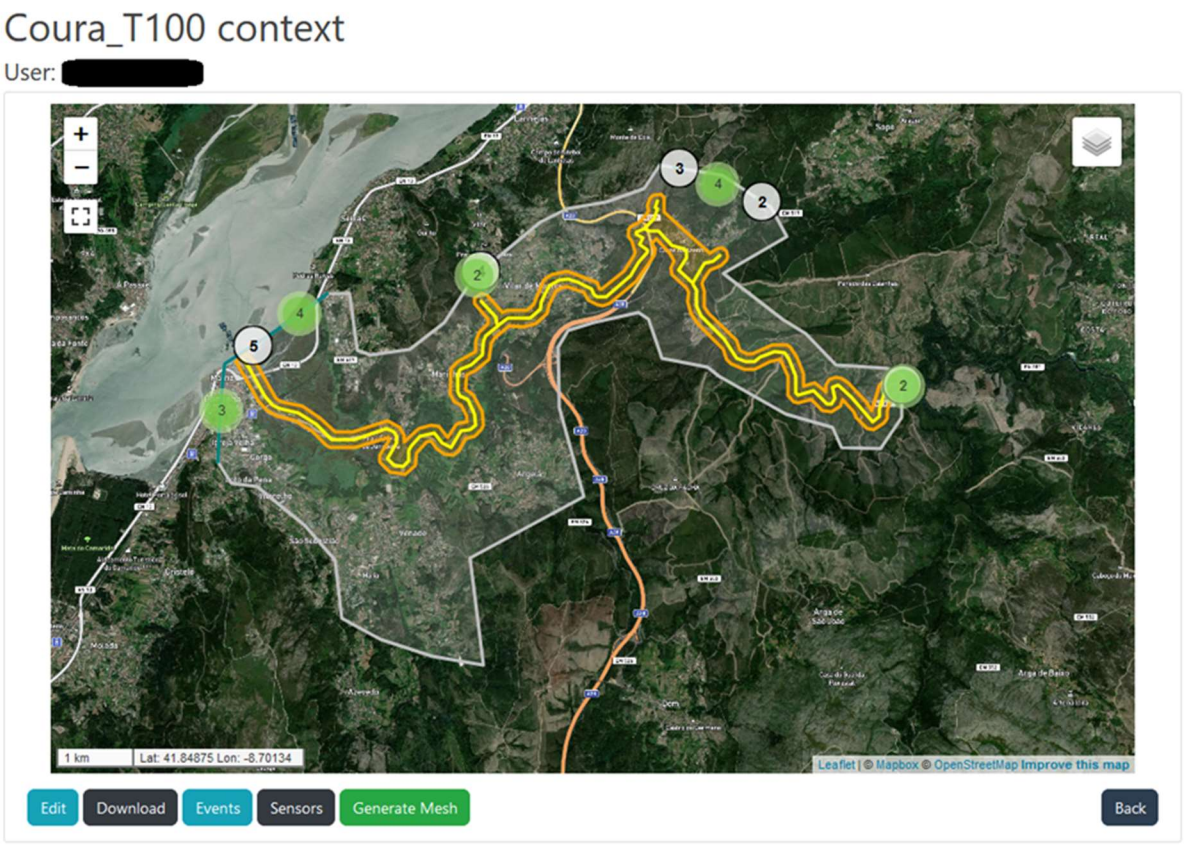

Figure 29 - Context Detail Page in RCP

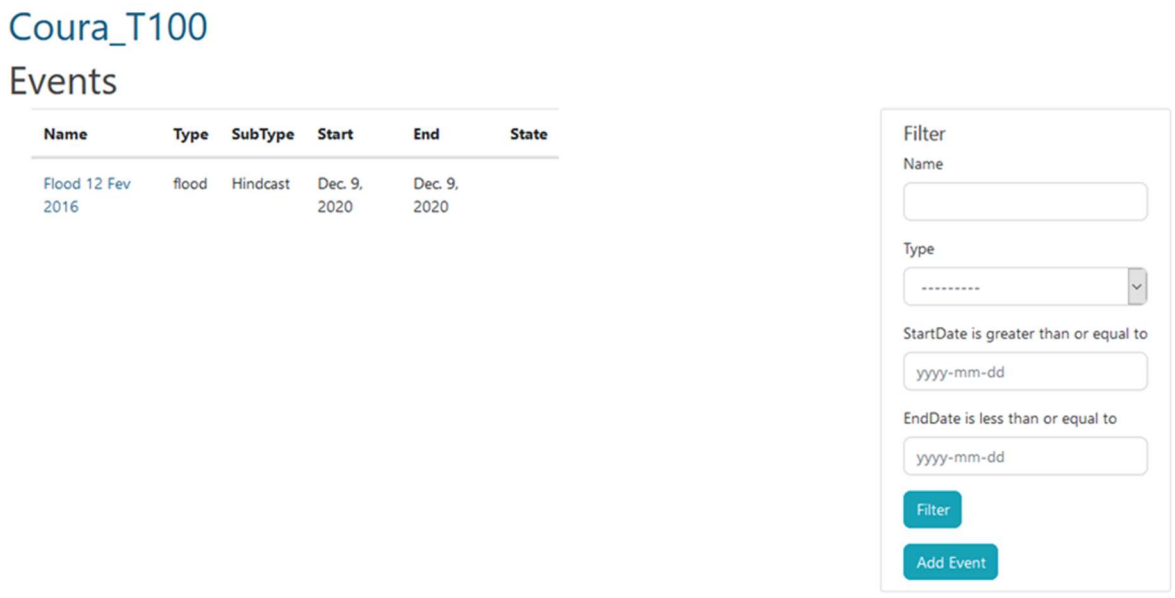

Figure 30 - Event List Page in RCP

Appendix D

Xtend main script for the generation of User Interfaces

```
package org.itlingo.asl.generator.ASL_From_UseCases

import java.util.ArrayList
import java.util.List
import org.itlingo.asl.asl.DataAttributeTypeOriginal
import org.itlingo.asl.asl.DataEntityCluster
import org.itlingo.asl.asl.System
import org.itlingo.asl.asl.UseCase
import org.itlingo.asl.asl.UseCaseTypeOriginal
import org.itlingo.asl.asl.DataEntity

class ASLMainContainers {
  def static compile(UseCase usecase, ArrayList<UseCase> usecases, System system){

    var List<String> all_editable_attributes = newArrayList
    var List<String> readonly_attributes = newArrayList
    var List<String> visualization = newArrayList
    var List<String> order = newArrayList
    val ArrayList<String> visibleTypes = new ArrayList();

  visibleTypes.add("String");visibleTypes.add("Integer");visibleTypes.add("Decimal");visibleTy
pes.add("Boolean");

  visibleTypes.add("Currency");visibleTypes.add("Date");visibleTypes.add("Time");visibleTypes.
add("DateTime");

    var main_ent = ""
    if ((usecase.dataEntity instanceof DataEntityCluster)){
      main_ent = (usecase.dataEntity as DataEntityCluster).main.name

      for( attribute : (usecase.dataEntity as DataEntityCluster).main.attributes){
        if (attribute.constraint!==null && attribute.constraint.isReadOnly=='isReadOnly')
          readonly_attributes.add(0, attribute.name)
        else
          all_editable_attributes.add(0, attribute.name)
      }

      for( attribute : (usecase.dataEntity as DataEntityCluster).main.attributes){
        if( (attribute.type  instanceof DataAttributeTypeOriginal)){
          var att_type = (attribute.type as DataAttributeTypeOriginal).type
          if (visibleTypes.contains(att_type)){
            visualization.add(0, attribute.name);
            order.add(0,attribute.name);
          }
        }
      }
    }
    else{
      if ((usecase.dataEntity instanceof DataEntity)){
      main_ent = (usecase.dataEntity as DataEntity).name

      for( attribute : (usecase.dataEntity as DataEntity).attributes){
        if (attribute.constraint!==null && attribute.constraint.isReadOnly=='isReadOnly')
          readonly_attributes.add(0, attribute.name)
        else
          all_editable_attributes.add(0, attribute.name)
      }

      for( attribute : (usecase.dataEntity as DataEntity).attributes) {
        if( (attribute.type  instanceof DataAttributeTypeOriginal)){
          var att_type = (attribute.type as DataAttributeTypeOriginal).type
          if (visibleTypes.contains(att_type)){
            visualization.add(0, attribute.name);
            order.add(0,attribute.name);
          }
        }
      }
```

```
       }

     }
'''
Package p_«system.name»

Import p_«system.name».«system.name»
Import p_«system.name».«system.name».*

System UI_«system.name» : Application [ ]

//UIContainer aligned with «usecase.name»
//MAIN WINDOW UI

UIContainer uiCt_«main_ent» : Window [

   component uiCo_«main_ent»List : List: List_Table [
     isScrollable
     dataBinding «main_ent» [
       visualizationAttributes «FOR v : visualization» «main_ent».«v»,«ENDFOR»
       sortAttributes «FOR v : visualization» «main_ent».«v»,«ENDFOR»
       //orderBy «FOR o : order»«main_ent».«order.get(0)» «ENDFOR» DESC
     ]

   part uip_checkselect: Field: Field_Input : WFC_CheckBox
   ]

   component uiCo_Filter_«main_ent»: Details [
     dataBinding «main_ent» [
       filterAttributes «FOR v : visualization», «main_ent».«v»«ENDFOR»
     ]
   ]

   component uiCo_Search_«main_ent»: Details [
     dataBinding «main_ent» [
       searchAttributes «FOR v : visualization», «main_ent».«v»«ENDFOR»
     ]
   ]

   component uiCo_Actions: Menu [

     event uiev_create "Create «main_ent»" : Submit: Submit_Create [navigationFlowTo
uiCt_«main_ent»Creator]
     event uiev_delete "Delete «main_ent»" : Submit: Submit_Delete
     event uiev_update "Edit" : Submit: Submit_Update [navigationFlowTo
uiCt_«main_ent»Creator]
   ]

]
«var usecase_type = ""»
«IF usecase.type instanceof UseCaseTypeOriginal»
   //UC Type: «usecase_type = (usecase.type as UseCaseTypeOriginal).type»
«ENDIF»

«IF usecase_type == "EntitiesSearch" || usecase_type =="EntitiesManage"»
//UI FOR SEARCH

component uiCo_Search: Details [
   part uip_search: Field: Field_Output: WFC_Label [Text defaultValue "Search:"]
   part uip_search_«main_ent»: Field: Field_Input : WFC_Text [Text defaultValue "Search
«main_ent»"]
   event uiev_search "Search" : Submit: Submit_Ok
]
«ENDIF»
«IF usecase_type == "EntityDelete" || usecase_type =="EntitiesManage" || usecase_type
=="EntitiesBrowse"»
// UI FOR DELETE

UIContainer uiCt_Delete_Warning : Window  [
   component uiCo_DeleteWarning : Dialog : Dialog_Warning  [

     part uip_Delete_Warning_Text: Field: Field_Output [Text defaultValue "Are you sure you
want to delete «main_ent»?" ]
     event uiev_delete : Submit: Submit_Delete [navigationFlowTo uiCt_«main_ent»]
     event uiev_cancel : Submit: Submit_Cancel [navigationFlowTo uiCt_«main_ent»]
```

```
     ]
   ]

UIContainer uiCt_Delete_Confirmation : Window : Window_Modal [
   component uiCo_Delete_Message : Dialog : Dialog_Info [
      part uip_Delete_Confirmation_Text: Field: Field_Output [Text defaultValue "«main_ent»
Deleted" ]
      event uiev_confirm : Submit : Submit_Ok [navigationFlowTo uiCt_«main_ent»]
     ]
]
«ENDIF»
«IF usecase_type == "EntityRead" || usecase_type =="EntitiesManage" || usecase_type
=="EntitiesBrowse"»

UIContainer uiCt_«main_ent»Reader : Window: Window_Modal[

   component uiCo_Edit«main_ent» : Form : Form_Simple [
   «FOR attribute : all_editable_attributes»
               part uip_«attribute»Label  : Field: Field_Output : WFC_Label [Text defaultValue
"«attribute»"]
               part uip_«attribute»Value : Field: Field_Input : WFC_Text[Text defaultValue
"Editable Value"]
        «ENDFOR»
        «FOR attribute : readonly_attributes»
               part uip_«attribute»Label  : Field: Field_Output : WFC_Label [Text defaultValue
"«attribute»"]
               part uip_«attribute»Value : Field: Field_Output : WFC_Text[Text defaultValue
"Not changeable Value"]
        «ENDFOR»
   ]
component uiCo_«main_ent»NavigationButtons : Details [

   event uiev_beginning_«main_ent» "Beginning":   Submit: Other [navigationFlowTo
uiCt_«main_ent»Reader]
   event uiev_previous_«main_ent» "Previous":     Submit: Other [navigationFlowTo
uiCt_«main_ent»Reader]
   event uiev_next_«main_ent»  "Next":       Submit: Other [navigationFlowTo
uiCt_«main_ent»Reader]
   event uiev_end_«main_ent» "End":           Submit: Other [navigationFlowTo
uiCt_«main_ent»Reader]
   event uiev_back "Back":               Submit: Other [navigationFlowTo uiCt_«main_ent»]
]

]
«ENDIF»
«IF usecase_type == "EntityCreate" || usecase_type == "EntityUpdate" || usecase_type
=="EntitiesManage" || usecase_type =="EntitiesBrowse"»
//UI FOR CREATE/UPDATE

   UIContainer uiCt_«main_ent»Creator : Window : Window_Modal[
   component uiCo_Edit«main_ent» : Form : Form_Simple [
        «FOR attribute : all_editable_attributes»
               part uip_«attribute»Label  : Field: Field_Output : WFC_Label [Text defaultValue
"«attribute»"]
               part uip_«attribute»Value : Field: Field_Input : WFC_Text[Text defaultValue
"Editable Value"]
        «ENDFOR»
        «FOR attribute : readonly_attributes»
               part uip_«attribute»Label  : Field: Field_Output : WFC_Label [Text defaultValue
"«attribute»"]
               part uip_«attribute»Value : Field: Field_Output : WFC_Text[Text defaultValue
"Not changeable Value"]
        «ENDFOR»
        event uiev_create "Create «main_ent»"  : Submit : Submit_Create [navigationFlowTo
uiCt_«main_ent»]
        event uiev_cancel "Cancel «main_ent»"  : Submit : Submit_Cancel [navigationFlowTo
uiCt_«main_ent»]
        ]
   ]

   «ENDIF»
'''

}

}
```

References

1. Ousterhout, J. K.: A philosophy of software design. Yaknyam Press (2018).

2. Martin, Robert C.: Clean Architecture: A Craftsman's Guide to Software Structure and Design. 1st edition. Prentice Hall, Upper Saddle River (2017).

3. Al-Fedaghi, S.: Developing Web Applications. International Journal of Software Engineering and Its Applications (2011).

4. Ferreira, D., Silva, A. R.: RSLingo: An Information Extraction Approach toward Formal Requirements Specifications. In: 2nd IEEE International Workshop on Model-Driven Requirements Engineering. IEEE Computer Society (2012).

5. Sommerville, I.: Software engineering. 9th edition. Pearson, Boston (2011).

6. Shah, Tejas & Patel, S.: A Review of Requirement Engineering Issues and Challenges in Various Software Development Methods. International Journal of Computer Applications (2014).

7. Silva, A. R.: Rigorous Specification of Use Cases with the RSL Language. In Proceedings of the Information Systems Development (ISD'2019) Conference. AIS (2019).

8. Silva A.R.: Linguistic Patterns and Linguistic Styles for Requirements Specification (I): An Application Case with the Rigorous RSL/Business-Level Language. In Proceedings of EuroPLOP'2017, ACM (2017).

9. Silva A.R., Paiva, A. C. R., Silva, V. R.: A Test Specification Language for Information Systems based on Data Entities, Use Cases and State Machines. In Model-Driven Engineering and Software Development, Communications in Computer and Information Science, vol 991, Springer (2019).

10. Paiva, A. C. R., Maciel, D., Silva, A. R.: From Requirements to Automated Acceptance Tests with the RSL Language. In Evaluation of Novel Approaches to Software Engineering, Communications in Computer and Information Science, vol 1172, Springer (2020).

11. G. R. Pereira, Técnicas Avançadas de Modelacão e Producão Semi-Automática de Aplicações Web Responsivas, MSc Dissertation, Instituto Superior Técnico, Universidade de Lisboa, 2019

12. OMG, Interaction Flow Modeling Language Specification Version 1.0, https://www.omg.org/spec/IFML/1.0/, last accessed 2020/04/25.

13. M. G. B. Gonzalez, RiverCure Portal: Collaborative GeoPortal for Curatorship of Digital Resources in the Water Management Domain "" MSc Thesis, Instituto Superior Técnico, Universidade de Lisboa, 2020

14. K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," J. Manag. Inf. Syst., vol. 24, no. 3, 2007. 15. Schmidt, D.: Model-driven engineering. IEEE Computer. 39, 41-47 (2006).

16. Silva, A. R.: Model-driven engineering: A survey supported by A unified conceptual model. Computer Languages, Systems & Structures, 43 ,139-155, Elsevier (2015).

17. A.M. Davis, “Software Requirements: Objects, Functions and States”, Englewood Cliffs, 1993

18. Kraeling M, Tania L., Software Engineering for Embedded Systems (Second Edition), 2019

19. Ian Sommerville, Problems with Natural Language, 2008

20. Markus Voelter, DSL Engineering Designing, Implementing and Using Domain Specific Languages

21. https://martinfowler.com/books/dsl.html (last accessed on 31/12/2018)

22. Arie van Deursen and Paul Klint, Domain-Specific Language Design Requires Feature Descriptions

23. Tomaž Kosar, Nuno Oliveira, Marjan Mernik, Maria João Varanda Pereira, Matej Črepinšek, Daniela da Cruz, and Pedro Rangel Henriques, University of Maribor, Faculty of Electrical Engineering and Computer Science, Smetanova 17, 2000 Maribor, Slovenia {Comparing General-Purpose and Domain-Specific Languages: An Empirical Study

24. Rolf Schwitter, Centre for Language Technology, Macquarie University, Controlled Natural Languages for Knowledge Representation

25. Brambilla, M., Fraternali, P.: Interaction flow modeling language: Model-driven UI engineering of web and mobile apps with IFML (2014).

26. Stefano C., Fraternali P., Bongio A.: Web Modeling Language (WebML): A modeling language for designing Web sites. Computer Networks. 33. 137-157. 10.1016/S1389- 1286(00)00040-2 (2000).

27. https://www.eclipse.org/Xtext/ (last accessed on 27/12/2020)

28. https://www.eclipse.org/xtend/documentation/ (last accessed on 27/12/2020)

29. https://blogs.itemis.com/en/building-domain-specific-languages-with-xtext-andxtend (last accessed on 30/6/2020)

30. https://www.eclipse.org/xtend/documentation/202_xtend_classes_members.html/ (last accessed on 30/12/2020)

31. https://www.tiobe.com/tiobe-index/ (last accessed on 27/12/2020)

32.https://www.business2community.com/tech-gadgets/pros-and-cons-of-django-web-framework-for-app-development-02330165 (last accessed on 27/12/2020)

33. https://data-flair.training/blogs/django-advantages-and-disadvantages/ (last accessed on 27/12/2020)

34. Vincent, W.S.: Django for beginners: build websites with Python & Django (2018)

35. https://docs.djangoproject.com/en/2.2/ref/templates/language/ (last accessed on 27/12/2020)

36. https://getbootstrap.com/ (last accessed on 27/12/2020)

37. https://jquery.com/ (last accessed on 27/12/2020)

38. https://reactjs.org/ (last accessed on 27/12/2020)

39. Vincent, W.S.: Django for APIs: Build web APIs with Python & Django (2020)

40. D. Rubio: Beginning Django: Web Application Development and Deployment with Python. F. Bahia, Ensenada, Baja California, Mexico (2017).

41. Pinkham, A.: Django unleashed. 1st edition. Pearson, Indiana (2016).

42. https://docs.djangoproject.com/en/2.2/ref/contrib/admin (last accessed on 27/12/2020)

43. Mendix Evaluation Guide, https://www.mendix.com/evaluation-guide, last accessed 2020/4/26.

44. OutSystems Evaluation Guide, 16 https://www.outsystems.com/evaluation-guide, last accessed 2020/4/26.

45. López-Landa, R., Noguez, J., Guerra E., Lara, J.: EMF on rails, In: ICSOFT 2012 - Proceedings of the 7th International Conference on Software Paradigm Trends. 273-278 (2012).

46. Stefano Ceri; Piero Fraternali & A. Bongio (May 2000). "Web modelling language (WebML): A modelling language for designing Web sites". Proceedings of 9th International World Wide Web Conference, Amsterdam, 2000

47. Silva A.R., Saraiva J., Silva R., Martins C.: XIS – UML Profile for eXtreme Modeling Interactive Systems, In: Proceedings of the MOMPES 2007, IEEE Computer Society (2007). 48. Ribeiro, A., Silva, A. R.: XIS-Mobile: A DSL for Mobile Applications, Proceedings of the 29th Annual ACM Symposium on Applied Computing (2014).

49. Ribeiro, A., Silva, A. R.: Evaluation of XIS-Mobile, a Domain Specific Language for Mobile Application Development, Journal of Software Engineering and Applications, 7(11), pp. 906-919 (2014).

50. Seixas, J., Ribeiro, A., Silva, A. R.: A Model-Driven Approach for Developing Responsive Web Apps", Proceedings of ENASE'2019, SCITEPRESS (2019).

51. https://djangosuit.com/ (last accessed on 27/12/2020)

52. https://docs.djangoproject.com/en/2.2/topics/class-based-views/ (last accessed on 27/12/2020)

53. https://django-filter.readthedocs.io/en/stable/ (last accessed on 27/12/2020)

54. Rubio, D.: Beginning Django: Web Application Development and Deployment with Python. 1st edition. Apress, Ensenada, Baja California, Mexico (2017).

55. https://sparxsystems.com/products/ea/index.html (last accessed on 27/12/2020)

56. https://code.visualstudio.com/ (last accessed on 27/12/2020)

57. https://github.com/uctakeoff/vscode-counter (last accessed on 27/12/2020)

58. http://cloc.sourceforge.net/ (last accessed on 27/12/2020)

59. R. Gomes, M. Saramago, and R. Rodrigues, “SVARH - Sistema de Vigilância e Alerta deRecursos Hídricos,” Relatório Técnico, Instituto da Água, 2003.

60. http://www.prociv.pt/en-us/Pages/default.aspx (last accessed on 30/12/2020)

61. M. Saramago, “Redes de Monitorização Hidrometeorológicas,” Rev. Recur. Hídricos, Assoc.Port. dos Recur. Hídricos, vol. 38, no. 1, 2017.

62. https://www.postgresql.org/ (last accessed on 1/12/2020)

63. Pereira J. M. S., Classifying Geo-Referenced Photos and Segmenting Satellite Imagery for the Assessment of Flood Severity, MSc Dissertation , Instituto Superior Técnico, Universidade de Lisboa, 2019

64. Marques J. M. S., RiverCure Portal: Exploring Geographic features on context definition and integration with the histav hydrometic tool, MSc Dissertation, Instituto Superior Técnico, Universidade de Lisboa, 2020

65 D. A. S. Conde, “High-Performance Modelling of Tsunami Impacts on Built Environments,” PhD Thesis, Universidade de Lisboa, Instituto Superior Técnico, 2018.

66. https://www.paraview.org/ (last accessed on 15/12/2020)

67. https://www.paraview.org/web/ (last accessed on 15/12/2020)

68. Montero, S., Díaz, P., Aedo, I.: From requirements to implementations: A model-driven approach for web development. In: EJIS. 16. 407-419. 10.1057/palgrave.ejis.3000689 (2007).

69. https://genio.quidgest.com/ (last accessed on 27/12/2020)

70. Silva, A. R., Savic D.: Linguistic Patterns and Linguistic Styles for Requirements Specification: Focus on Data Entities. INESC-ID Technical Report, 2020.

71. Silva, A. R.: Linguistic Patterns and Linguistic Styles for Requirements Specification: Focus on Use Cases and Scenarios. INESC-ID Technical Report, 2020.

72. Gamito I, Silva, A. R.: From Rigorous Requirements and User Interfaces Specifications into Software Business Applications. QUATIC, 2020.