From Rigorous Requirements and User Interfaces Specifications into Software Business Applications

Ivo Miguel Torrado Gamito

Instituto Superior Técnico, Universidade de Lisboa, Portugal

ivo.gamito@tecnico.ulisboa.pt

Abstract. Software applications have been developed with multiple programming languages (specific software libraries and frameworks) and deployed on various software and hardware infrastructures. This paper introduces and discusses the ASL language (short for "Application Specification Language") that combines constructs from two previous languages: ITLingo RSL and IFML. ASL specifications are strict and rigorous sentences that allow us 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. Furthermore, the proposed approach includes model-to-model and model-to-code transformations that may considerably improve the quality and productivity of both the requirements specification and the development of software applications.

Keywords: Requirements Engineering, Model-driven engineering, Web engineering, ITLingo RSL, IFML, ITLingo ASL

1 Introduction

Currently, developers use expressive programming languages, software libraries and frameworks that help them develop a multitude of software applications. However, developers have to master details of these tools and technologies which [1] 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 [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. 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]. Also, RE has also been crucial for the success of a project and it has dealt with socio-technical challenges like the adoption of elicitation techniques, communication difficulties, and or with conflicting and ambiguous requirements [6].

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.

We propose in this paper an approach to improve the RE process by mitigating some of its problems, namely in what concerns the specification and validation of requirements. This is also a model-driven approach. i.e., an approach that considers models not just documentation artifacts, but also central artifacts in the software engineering area, allowing automatic creation of software applications starting from those models. Model-driven engineering (MDE) involves the adoption of languages and transformation engines to address the diversity and complexity of software platforms and frameworks [7]. In the scope of MDE, we consider a model as an abstraction of a system often used to replace the system under study [8]. 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 (programing language) artifacts, or it can be directly executed by some interpretation engine [8].In this context, we introduce and discuss the ASL specification language ("Application Specification Language") that combines constructs from two languages (further details in the next section): ITLingo RSL [4,9,20,23] and IFML [10]. 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, based on the concepts found in modeling languages like IFML. ASL gathers characteristics and advantages from both RSL and IFML. Likewise RSL, and differently from IFML (and 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"). Also, to the rigorous and systematic specification of software applications, we show that it is possible to take advantage of these specifications to semi-automatically generate software applications following an MDE approach: this means that with appropriate tools, an ASL user can create web applications, which can be generated through automatic transformations techniques, from ASL rigorous specifications.

2 Background

This section briefly introduces ITLingo RSL and IFML languages, in which the ASL is based on.

2.1 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 [9,20,23,24].

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, behavior, passive structure, requirements, tests, relations and sets, and others [9,23]. 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.

2.2 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 behavior of the applications as perceived by end-users [10].

IFML brings benefits to the development process of application front-ends, namely [11]: supports the specification of application front-ends with different perspectives (the connection with the business logic, the data model, and the graphical presentation layer); 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 [22], as well as by other experiences in the Web modeling field [10]. IFML intends to solve a problem mentioned above 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 [10]: 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 4 (left) shows a simple example of an IFML model.

3 ASL Language

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 also be classified as "business applications, in which data is a core asset, and support several business activities, like planning, forecasting, control, coordination, decision making and operational activities [17]. Popular classes of software business applications are e-commerce, ERP (enterprise resource planning), CRM (customer relationship management), SCM (supply chain management).

This section introduces the ASL architecture. This discussion is supported by a simple running example named "MyTinyBlog" application, described as follows: « *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 1 and 2 illustrate some models of the MyTinyBlog application: the domain model and the use-cases model. The main feature of this application involves managing blog posts through typical create, read, update and delete (CRUD) operations.

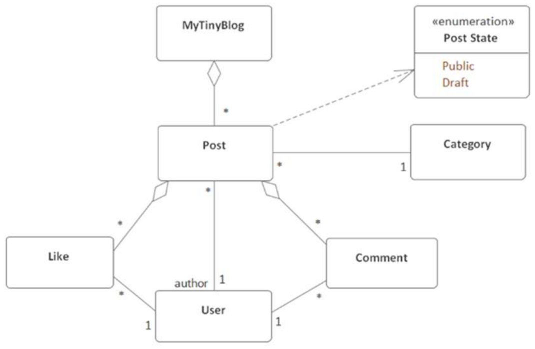


Fig. 1. MyTinyBlog data model (UML class diagram)

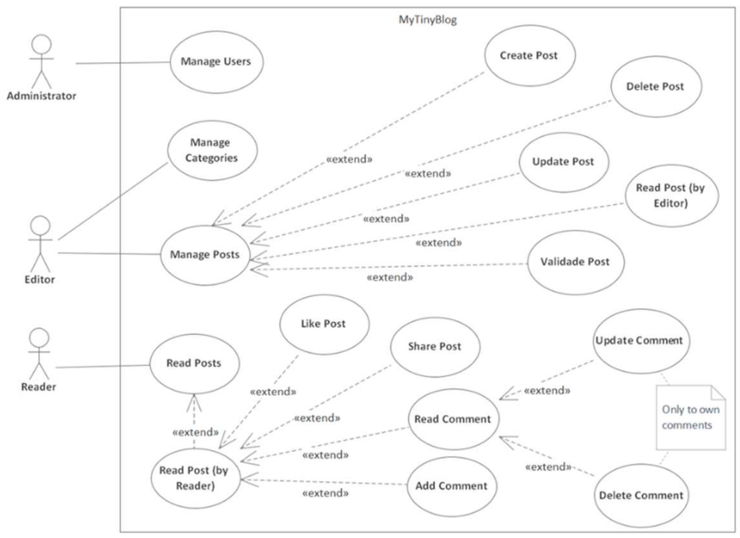


Fig. 2. MyTinyBlog use cases model (UML use cases diagram)

3.1 Data Entities

ASL adopts and extends the definition of the DataEntity construct as defined initially in RSL [9,20]. 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 [9]. A DataEntity can be classified by type and subtype. The types are the following: (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).

In the MyTinyBlog example, we define the following data entities, as also suggested in Figure 1: Blog, Category, User, Blog Post, Comment and Like (see Spec. 2).

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

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

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": Master [
   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": Reference [
   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":  Reference [
   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 2. Specification of MyTinyBlog's data entities (in ASL)

After defining the data entities, DataEntityClusters can be defined. A DataEntityCluster construct denotes a cluster 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", in the ec_PostComment cluster, is used as an extended property to influence model-to-model or model-to-code transformations (in what respect the UI definition of the application).

```
//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 3. Specification of data entity clusters (in ASL)

3.2 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 [9]. 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 example (see Spec. 4), we define the ContextActor "Blog Editor" that creates and manages blog posts. The use case "Manage Blog Posts" is initiated by the "Blog Editor" that involves the management of data cluster "Blog Posts" (ec_Post) with CRUD actions.

```
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_Editor
   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 4. Specification of MyTinyBlog's actors and use cases (in ASL)

3.3 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 Figure 4.

Table 1. Supported types used for the UIViewComponent definition

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

4 The ASL-based approach

Figure 3 suggests the approach proposed to systematically and rigorously define software applications based on the ASL language. This approach includes the possibility of automatically generating the software application for a specific software platform.

The proposed approach consists in 6 main tasks, represented in Figure 3. Task 1 starts with a developer specifying the data and use cases models. Then, Task 2 automatically validates that partial model. If this model is valid, the ASL may run tool support may run model-to-model transformations to generate ASL UI specifications (Task 3) automatically. Then, in the Task 4, the developer can still add or change 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.

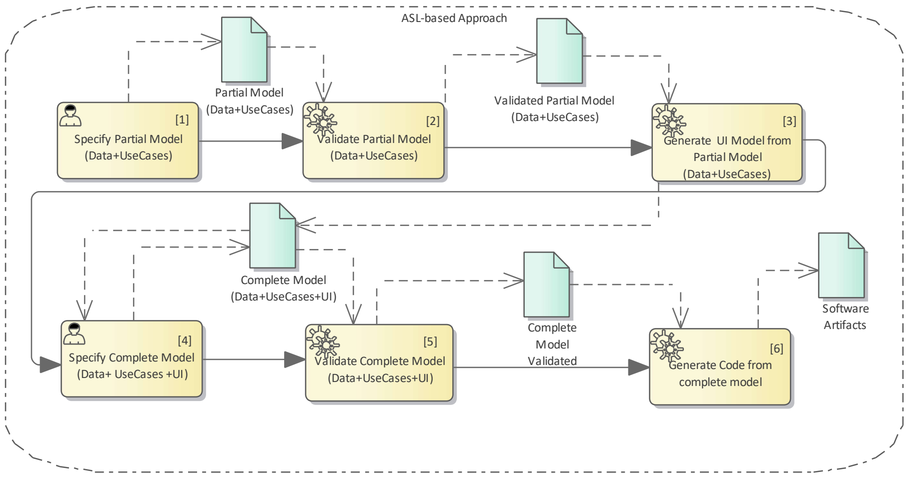

ASL-based Approach
Partial Model (Data+UseCases)
Validated Partial Model (Data+UseCases)
[1] Specify Partial Model (Data+UseCases)
[2] Validate Partial Model (Data+UseCases)
[3] Generate UI Model from Partial Model (Data+UseCases)
Complete Model (Data+UseCases+UI)
Complete Model Validated
Software Artifacts
[4] Specify Complete Model (Data+ UseCases +UI)
[5] Validate Complete Model (Data+UseCases+UI)
[6] Generate Code from complete model

Fig. 3. ASL-based proposed approach

4.1 Model-to-model transformation

The proposed approach follows an idea initially introduced with the XIS approach [12]: 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 (based on the XIS terminology), and then the User Interfaces views are automatically generated based on model-to-model (M2M) transformations and a predefined set of UI patterns [12].

We integrate that "smart approach" to the ASL approach, which allows to generate UI specifications, as referred above in Task 3. These generated ASL files include UI specifications that depend on the data entities and use cases previously defined. For instance, considering the use-case defined in Spec.4 (i.e., use case "Manage Blog Posts"(uc_1_ManagePost)), it generates UI elements to support CRUD actions of posts. The "databinding" mentions the e_Post entity. Features or actions like Listing, filtering and searching of e_Post can be then manually customized.

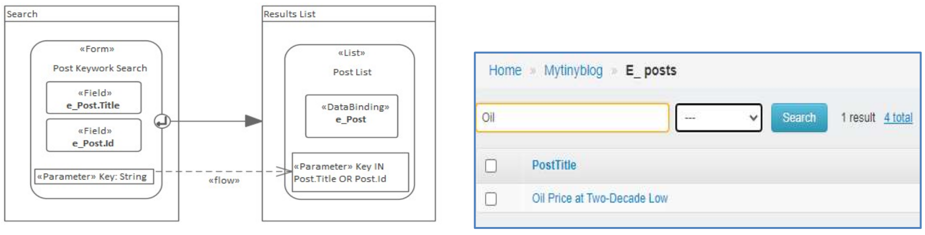


Fig. 4. Search posts: IFML model (left) and UI (right)

```
//Search
component uiCo_Search_e_Post: Details [
   dataBinding e_Post [searchAttributes e_Post.Title, e_Post.Id ]]
```

Spec 5. Generated specification for searching on MyTinyBlog (in ASL)

As suggested in Spec. 5 and Figure 4, the specification of that UI list table can be subject of further customization, like the definition of data attributes can be considered for filter and search features, or we can even customize the properties of each data attribute.

4.2 Model-to-code transformation

The transformation referred above as Task-3 generates ASL UI specifications, but the target software application (e.g., MyTinyBlog) is not yet developed and deployed. However, a 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 have developed model-to-code transformations into the Django web framework. Django is an open-source high-level Python Web framework that encourages rapid development and clean, pragmatic design [13,23].

```
#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')
```

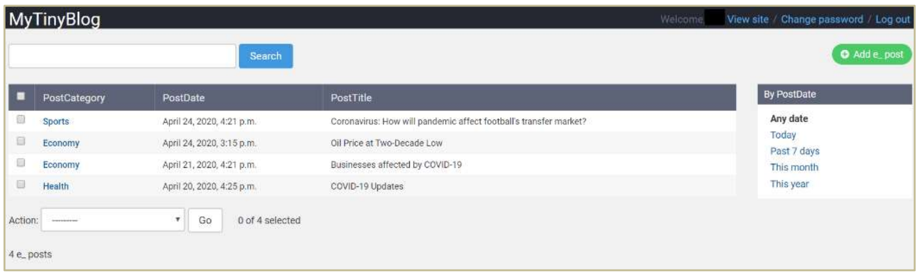
Spec 6. Generated Django model of the MyTinyBlog application (in Python)

As a simple example of these model-to-code transformations, Spec. 6 illustrates the corresponding Django data model for MyTinyBlog. This code is generated mainly from the data entities defined in ASL (see Spec.2). This transformation generates the file "models.py", which takes into account 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 still add more information (by default, Django uses SQLite database to store the data [13]).

Users (who were given permissions) can create, read, update or delete the blog posts using the Django admin site [13]. This site reads metadata from the models and provides a simple 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 replace it with new settings; It shall contain the models to be registered, and other constraints generated from the ASL specifications files (spec. 5). These options are visible in the generated application, as suggested in Figure 5.



	PostCategory	PostDate	PostTitle
☐	Sports	April 24, 2020, 4:21 p.m.	Coronavirus: How will pandemic affect football's transfer market?
☐	Economy	April 24, 2020, 3:15 p.m.	Oil Price at Two-Decade Low
☐	Economy	April 21, 2020, 4:21 p.m.	Businesses affected by COVID-19
☐	Health	April 20, 2020, 4:25 p.m.	COVID-19 Updates

0 of 4 selected

4 e_ posts

Fig. 5. MyTinyBlog posts list

4.3 Actors and Permissions

The implemented model-to-code transformations can also speed up the process of managing users and permissions (see Table 2 for some concepts mapping between ASL and Django). Python interpreter allows to create groups and assign permissions to users. However, Django Admin provides a built-in authentication system that allows the same features through a simple, intuitive interface [14].

In the MyTinyBlog application, the blog administrator oversees those 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 and users in the database. This script also assigns user groups different permissions. To quickly validate the authorization features, this script adds one user to each user group. All these settings can be later directly managed by a superuser using the Django admin site. (If logged in as a superuser, the user can create, edit, and delete any object; he can as well modify groups/permissions [14]).

Table 2. 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 7. Example of generated Django roles script of the MyTinyBlog application (in Python).

5 Related Work

Some approaches and tools have either improved and accelerated how the community has produced software applications, either developed by the industry (e.g., Mendix, Outsystems, or WebRatio) or by research settings (e.g., EMF on Rails, ADM, XIS or ITLingo RSL).

Mendix 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 [15]. Mendix builds a wide range of transactional, event-driven, and adjacent applications for all kinds of industries [15]. In Mendix perspective, it is becoming harder to keep up to date with the evolving number of programming tools and languages across the spectrum [15]. 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 [15].

OutSystems is another commercial platform for low-code rapid application development with advanced capabilities for enterprise mobile and web apps [16]. Starting in 2001, OutSystems recognized that a vast majority of software projects were failing, due to multiple reasons. Therefore, OutSystems software is an integrated development environment that covers the entire development lifecycle, namely: development, quality assurance, deployment, monitoring and management [16].

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.

EMF on Rails proposes an approach that combines MDE with automation frameworks for web development like Spring Roo [18]. 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". Like ASL, it accelerates the generation of CRUD operations on data models [18]. 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 [21]. 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 [21].

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 [3]. 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 front-end 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 [11].

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 [12]. XIS intends to reduce costs and improve the fulfillment 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 [25,26,27]. 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. 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.

6 Conclusion

This paper discusses a new approach that combines the disciplines of requirements engineering and web engineering. This approach intends to address the followings issues: How to better specify requirements and business applications' (user interfaces) in an integrated way and how to increase the productivity of developers by automatizing the production of artifacts like technical documentation and software code.

We discuss some existing solutions, namely those mostly related to the RSL and IFML languages, in which the ASL language 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 interface elements of the applications. We show that this language can be combined with tools that support both 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 the discussion with a simple but effective example, considering a popular class of web applications (i.e., a Blog application) on top of a popular Python-based framework (the Django framework).

Future research shall consider improving the customization of either the specification and generation of the business applications and shall specify and develop multiple cases studies. The integration with other popular software (e.g., NodeJS, JavaScript frameworks, .NET) and low-code frameworks (e.g., Mendix, OutSystems or Genio [19]) can also be considered as they can bring more flexibility to this solution. This research shall discuss how to consider how to deal with cross-cutting quality attributes such as availability, performance, usability or security.

Acknowledgments

Work supported by funds under FCT UID/CEC/50021/2019 and 02/SAICT/2017/29360.

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. Schmidt, D.: Model-driven engineering. IEEE Computer. 39, 41-47 (2006).
8. Silva, A. R.: Model-driven engineering: A survey supported by A unified conceptual model. Computer Languages, Systems & Structures, 43 ,139-155, Elsevier (2015).
9. Silva, A. R.: Rigorous Specification of Use Cases with the RSL Language. In Proceedings of the Information Systems Development (ISD'2019) Conference. AIS (2019).
10. OMG, Interaction Flow Modeling Language Specification Version 1.0, https://www.omg.org/spec/IFML/1.0/, last accessed 2020/04/25.
11. Brambilla, M., Fraternali, P.: Interaction flow modeling language: Model-driven UI engineering of web and mobile apps with IFML (2014).
12. 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).
13. Pinkham, A.: Django unleashed. 1st edition. Pearson, Indiana (2016).
14. Rubio, D.: Beginning Django: Web Application Development and Deployment with Python. 1st edition. Apress, Ensenada, Baja California, Mexico (2017).
15. Mendix Evaluation Guide, https://www.mendix.com/evaluation-guide, last accessed 2020/4/26.
16. OutSystems Evaluation Guide, 16 https://www.outsystems.com/evaluation-guide, last accessed 2020/4/26.
17. Stair, R., Reynolds, G.: Fundamentals of Information Systems 9th Edition, Cengage Learning (2017).
18. 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).
19. Genio Plataforma, https://genio.quidgest.com/plataforma/, last accessed 2020/6/26.
20. 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).
21. 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).
22. 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).
23. 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).
24. 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).
25. Ribeiro, A., Silva, A. R.: XIS-Mobile: A DSL for Mobile Applications, Proceedings of the 29th Annual ACM Symposium on Applied Computing (2014).
26. 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).
27. Seixas, J., Ribeiro, A., Silva, A. R.: A Model-Driven Approach for Developing Responsive Web Apps", Proceedings of ENASE'2019, SCITEPRESS (2019).
28. Django, https://www.djangoproject.com/, last accessed 2020/6/25.