Rapid Production of Enterprise Applications in a Low-Code Environment: Comparing the ITLingo-ASL and PowerApps Metamodels

Patrícia Borges Vilão
Instituto Superior Técnico, Universidade de Lisboa
Lisboa, Portugal *patriciavilao@tecnico.ulisboa.pt*

Alberto Rodrigues da Silva
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
Lisboa, Portugal *alberto.silva@tecnico.ulisboa.pt*

Abstract

Low-code development platforms allow to reduce the time and resources required for developing business applications; thus, many companies are increasingly adopting them. However, they often use proprietary languages making it challenging to interoperate with other systems or switch to different low-code platforms, resulting in vendor lock-in situations. This research proposes to combine a model-driven approach based on rigorous requirements specifications defined in the ITLingo-ASL language with the Microsoft PowerApps technology to generate quasi-complete enterprise applications semi-automatically. This research analyses the ITLingo-ASL and Microsoft PowerApps metamodels, mainly focusing on concepts related to DataEntities, UI elements, Actors and Use cases to find similarities indicating that transforming one model is possible. It also pinpoints differences so that the ITLingo-ASL language can be extended to support software enterprise application specifications better.

Keywords: Requirements engineering, Model-driven engineering, Low-code platforms, ITLingo ASL, Microsoft PowerApps.

1. Introduction

Developing enterprise applications requires cross-skilled teams to translate stakeholder needs into objective requirements understandable to all team members, regardless of their backgrounds [17]. This process, known as requirements engineering (RE), helps mitigate communication challenges and prevent mistakes based on misinterpreting system requirements [9]. On the other hand, companies are increasingly using low-code development platforms like Quidgest Genio [27], OutSystems [32], and Microsoft PowerApps [24] to develop enterprise applications, as they contribute to high productivity and reduce production and maintenance costs, allowing organisations to adapt to new requirements more efficiently [21]. Low-code platforms also promote user empowerment through citizen development, enabling non-experts to contribute to the IT business [2]. However, low-code platform languages are less abstract than platform-independent specification languages, although they allow for more comprehensive requirement specifications than programming languages typically do [8].

Additionally, Model-Driven Engineering (MDE) considers models not only as documentation artefacts but as central artefacts in the field of Software Engineering, enabling the automatic creation of software applications directly from these models [2][14]. It is important to note that MDE has similar objectives as low-code development [2], which renders some discussion on their differences and to what extent work done in MDE can be combined with low-code development platforms [4]. ITLingo is a research initiative that has proposed new languages, tools, and techniques to assist its users in processes regarding project management and requirements engineering [18]. ASL ("Application Specification Language") is a language for the specification of software applications in a platform-independent way that is part of the ITLingo initiative [9]. The work of Gamito and Silva [9] demonstrated that ASL can be combined with tools that support model-to-model and model-to-code transformations;

thus, their approach can significantly enhance both the quality and productivity of defining requirements and developing applications. This work mentions the possibility of verifying if ASL could be suitable to support interoperability between the models developed with low-code development platforms as work to explore in the future. Furthermore, one of the most relevant challenges of low-code development platforms is the dependency on their vendors. This happens since many of these platforms store the code of their applications in an untraceable file format, which can only be interpreted on their platform [7]. Another limitation is the lack of interoperability typical of these platforms, creating friction between the migration and sharing of solutions between different low-code development platforms [15].

In this research, we aim to explore techniques combining model-driven tools with low-code platforms, as mentioned in the future work of Gamito and Silva [9]. Galhardo and Silva [8] discuss a model-driven approach that produces software business applications by combining rigorous requirements specifications (defined with the ITLingo ASL language) with a concrete low-code platform (Quidgest Genio [27]). This approach is supported by a series of ASL-based transformations: ASL2ASL transformation, ASL2Genio, and Genio2ASL, providing round-trip interoperability between ASL and Genio. The transformations are supported by the ASL generator that takes ASL specifications as input and performs model-to-model and model-to-code transformations to bootstrap the development of software business applications that comply with the original requirements specification.

Our work aims to explore the combination of ASL with another low-code platform: Microsoft PowerApps [24]. These two embedded developments could represent a promising step towards achieving interoperability between low-code platforms and, consequently, a higher level of independence from vendor-specific solutions. With this technique, software developers can quickly create platform-independent software application specifications mostly focused on these applications' business, data, and user interface aspects. By automatically transforming ASL specifications into particular low-code platform specifications, users may then use the wide range of functionality offered by these platforms.

The work discussed in this paper differs from the work of Gamito and Silva [9], since in theirs, they explored model-to-code transformations, and in this project, the focus is on model-to-model transformations. Although the work developed by Galhardo and Silva [8] also focuses on model-to-model transformations, it also varies from ours since the low-code platform they chose to work with was Quidgest Genio [27], and in our case, it is Microsoft PowerApps [24]. Another difference between that work and ours is that we intend to conduct tests with PowerApps users, where the users will follow a script that contains the tasks they need to perform and then answer a satisfaction questionnaire so that we can later analyse the data gathered. Moreover, when publishing an application in Genio, it goes through a code generation process [8], while PowerApps publishes the model in runtime. This means it is significantly faster and less costly to test applications generated in PowerApps than in Genio.

The structure of this paper is organised into six sections: Section 2 (Background) introduces the main concepts of domain-specific languages and low-code development platforms, with special attention on ITLingo and Microsoft PowerApps; Section 3 (Related Work) presents and discusses works in the scope of languages and architectural views, model-driven requirements specifications, and low-code development platforms interoperability; Section 4 (ASL and PowerApps Languages Comparison) compares aspects of ASL and MSPA languages in terms of data entity elements, user interface elements, and use cases; Section 5 (ASL-based transformations) describes the transformations proposed, based on the comparison made in Section 4; and Section 6 (Conclusion) presents the main conclusion of this work.

2. Background

This section introduces and discusses general concepts and technologies underlying this research, namely domain-specific languages and MDE. It also discusses ITLingo ASL, a specification language to define software business applications, and Microsoft PowerApps, a low-code enterprise application development platform.

2.1. Domain-Specific Languages and Model-Driven Engineering

A domain-specific language (DSL) can be a programming or executable specification language that provides expressive capacity focused on, and typically limited to, a single-issue area through appropriate notations and abstractions [22]. Some popular examples of DSLs are HTML (Hypertext Markup Language) and SQL (Structured Query Language). In contrast, a general-purpose language (GPL) is a programming language designed for a wide range of applications across different domains. GPLs are typically more complex than DSLs because they are designed to be flexible enough to handle a wide range of problems [22]. Additionally, model-driven approaches for requirements specification involve using a specific type of model or framework to represent and analyse the requirements for a system or product. These approaches help ensure that the requirements are well-defined, complete, and consistent and accurately reflect the stakeholders' needs and constraints. Several studies have been developed in this scope [16].

The use of domain-specific languages (DSLs) to represent variability in software systems was explored by Markus Voelter and Eelco Visser [23]. Variability refers to the ability of a system to be customised or configured to meet the specific needs and requirements of different users or stakeholders. The variability problem also appears in LCDPs; thus, we also intend to use a DSL to address it.

Bézivin et al. [1] discuss how MDE approaches can solve practical engineering problems using domain-specific modelling languages (DSLs). MDE approaches that use DSLs involve creating and manipulating domain-specific models at different levels of abstraction throughout the development process. These models specify the requirements for the system or product and can be transformed into implementation-level artefacts (such as code or documentation) using model-to-text transformations.

Considering the works mentioned above, in this project we will use DSLs, specifically focusing on DSLs for enterprise applications since DSLs can offer a higher level of abstraction and expressiveness for a specific domain than GPLs [11]. This way, it is easier and more natural for domain experts to create and understand models in that domain and, therefore quicker to write and modify, enabling the productivity of the developers' team [12].

2.2. Low-Code Development Platforms

Leading IT organisations are progressively introducing and promoting low-code development platforms (LCDPs), which are simple-to-use visual environments that allow citizen developers to construct their software systems even if they lack programming experience [2]. Popular examples of low-code development platforms are Quidgest Genio [27], OutSystems [32], Microsoft PowerApps [24], and Mendix [26]. Market research firms predict that LCDPs will become increasingly important for organisations. Additionally, the growing number of scientific publications in recent years indicates a rising interest from the academic community, mainly concentrating on their advantages and drawbacks [13].

According to Bock and Frank [2], very few low-code development capabilities are unique in and of themselves; novelty comes from how these aspects are combined and integrated. These platforms stand out because they combine several well-known and traditional system design elements in one environment, which reduces the effort required to carry out ordinary operations when creating business applications within specific, more or less constrictive frameworks.

2.3. ITLingo ASL

ITLingo ASL is an application specification language that combines features of two existing languages: ITLingo RSL [20] and Interaction Flow Modeling Language (IFML) [28]. ASL is of particular interest for this project when compared to other modelling languages such as UML and BPMN since the first is text-based and the others are not. The fact that ASL is a textual language allows a better manipulation of its concepts, which is essential in this work. ASL has a concrete textual syntax, which allows requirements definition systematically and rigorously, integrating software requirements with user interface aspects consistently [9]. The more relevant aspects of ASL for this project are:

(1) Data entities used to define domain concepts or information entities like goods, persons, or business transactions. It may specify attributes, foreign keys, and other data constraints [9]; (2) Use cases: a series of interactions between an actor (or actors) and the system under study that provide some value from the actor's perspective [9]; and (3) User interface elements: such as UI containers, UI components, and UI parts.

In the context of the ITLingo Initiative, the following software tools have been made available to aid in writing *SL specifications (precisely, ASL specifications) [8]. ITLingo-Studio, an Eclipse-based desktop program for meticulously generating *SL specifications, currently provides the finest capability for authoring ASL specifications [18]. An ITLingo-Cloud[1] version of the ITLingo-Studio has been created as current development. This technology serves as a collaboration platform for several organisations and projects. Compared to the desktop Studio version, it would be simpler and more efficient for users to maintain technical documentation [8, 18].

2.4. Microsoft PowerApps

Microsoft PowerApps (MSPA) is a cloud-based low-code platform that allows users to create custom business apps for various purposes, such as data collection, automation, and integration. This LCDP focuses on developing business apps without manual coding [10]. It is part of the Microsoft Power Platform [30] and works with other tools and services, including Microsoft Dataverse [31], Microsoft Power Automate [25], and Microsoft Power BI [29]. Microsoft Dataverse is a cloud-based data platform that stores and manages data for custom apps built with PowerApps. It is a fully managed, multi-tenant database service that allows users to create and store data in various formats, including tables, entities, and relationships [31]. Dataverse can be accessed through PowerApps [24], Power Automate [25], and other tools and platforms and is designed to be scalable, secure, and easy to use. Microsoft Power Automate [25] is a cloud-based service that allows users to automate business processes by creating workflows that trigger actions based on specific events or conditions. Microsoft Power BI [29] is a business analytics service that enables users to visualise and analyse data from various sources in a customisable and interactive way. With Power BI, users can connect to a wide range of data sources, such as spreadsheets and databases, and create interactive reports to gain insights into their data. The service includes various tools and features for data preparation, modelling, and analysis and supports collaboration and sharing of reports and dashboards with others in the organisation. Together, PowerApps [24], Dataverse [31], Power Automate [25], and Power BI [29] provide a powerful and flexible solution for businesses looking to streamline their operations and improve efficiency [30]. They can be used to build custom business apps that collect, store, and manage data, automate processes, and integrate with other applications and services.

Using PowerApps, the user can build two types of apps: canvas and model-driven. The key difference between a canvas app and a model-driven app is that a canvas app provides a blank canvas where you can design the user interface and add custom functionality. In contrast, a model-driven app is built on top of a specific data model and generates the user interface automatically based on that data model [34]. In this work, we will focus on canvas apps since they allow more flexibility in the definition of applications. When you export a canvas app from PowerApps, several files are created. The most relevant files are a .msapp file that contains the entire canvas app. It includes all of the app's screens, data sources, and other resources; One or more .png files containing screenshots of each screen. They provide a visual reference for the app's design; and one or more .json files containing metadata about the app, such as the app's name, description, and author.

[1] https://itlingocloud.herokuapp.com

3. Related Work

This section presents the related work involving the analysis of low-code development platforms and their interoperability mechanisms. Low-code platforms have been steadily gaining visibility, increasing their research interest.

Bragança et al. discuss the use of domain-specific languages (DSLs) to support software product line (SPL) engineering in low-code platforms [3]. SPL engineering is a software development approach that involves creating and maintaining a family of related software products that share a common core of features and functionality but can be customised or configured to meet the specific needs and requirements of different users or stakeholders [5]. Bragança et al. propose a model-to-model transformation, which takes as input a representation of the low-code model, such as a JSON file, and generates its corresponding low-code metamodel in a semi-automatic way [8].

In our proposal, the tool already has prior knowledge about the metamodel of the low code platform, meaning that the tool is not updated automatically, which implies constant maintenance. However, it is the fact that there is this prior knowledge that allows the transformation process to be automatic and not semi-automatic, like in Bragança et al.

As mentioned, our work intends to use the same conceptual model of transformations used in Galhardo and Silva's [8] work. For these transformations to occur, the low-code platform we are working with must have import and export mechanisms and provide some interoperability.

Table 1 presents a comparison between OutSystems [32], Mendix [26], Microsoft PowerApps [24], and Quidgest Genio [27] regarding their import and export mechanisms. The import/export mechanisms evaluate the presence of import and export capabilities of the LCDP, even if they differ. The specification of the elements that these platforms import/export is performed further below in Table 1. Open interoperability refers to the platform's ability to integrate with other software and systems. Finally, the format, content, and elements specify what each mechanism supports [8, 24, 26, 27, 32].

Table 1. Comparison between low-code development platforms regarding their import/export capabilities.

Capability	**OutSystems**	**Mendix**	**Microsoft PowerApps**	**Quidgest Genio**
Import/ Export	Yes	Yes	Yes	Yes
Open interoperability	-	-	Yes	Yes
Format	OutSystems Module (.oml)	Mendix Model Definition (.mpr)	Canvas App Package (.zip)	Genio model (.zip)
Contents	Encrypted XML (binary)	XML (text), JSON	.msapp, YAML, JSON (text)	XML (text)
Elements	Simple asset or snippet (e.g. small widget or a codding pattern)	Project definitions	App and Power Automate Flows	Project definitions

As summarised in Table 1, all the analysed LCDPs provide some import and export mechanisms. However, the main problem concerns interoperability since some LDPCs have restrictions regarding the file formats that can be imported and exported, making it very difficult or even impossible to interact with them in this way. Additionally, some of these platforms only provide interoperability within solutions of the same platforms, meaning that they can import and export different components or modules within the platform to work together seamlessly.

Regarding the elements each import/export mechanism supports, we see that these range from simple assets or snippets (as in OutSystems) to whole project definitions (as in Mendix, PowerApps, and Genio). A simple asset is a reusable component or function used when developing an application and can include UI components or functional modules such as authentication or data validation. A snippet is a small piece of code or a predefined set of actions

that can be effortlessly inserted into an application to accomplish a specific task, such as generating a PDF document or sending an email [32].

Mendix, PowerApps, and Genio support importing and exporting project definitions or apps. A project definition is a set of files that contain all the information required to open and work on a specific project. These files contain information such as the project name and settings and the location of the project files, providing a clear overview of the project structure and components.

4. ASL and PowerApps Languages Comparison

In our research, we have to work with an LDPC that would support third-party applications to interact with files that contain the specification of apps since we have to perform multiple transformations that generate files which need to be imported. We verify that Microsoft PowerApps meets the required conditions to be the second low-code platform to support interoperability with ITLingo ASL after the work of Galhardo and Silva [8], which used Quidgest Genio platform [27].

This section compares ASL and MSPA languages to align their concepts and better understand how the mapping between them will work. This comparison is done based on their metamodels and intends to pinpoint: (1) concepts that are defined in both languages so that the ASL2MSPA generator can transform them directly; and (2) concepts that are defined in MSPA but not in ASL so that extensions to ASL can be defined. This analysis focuses on data entities, user interfaces, actors and use cases, and related concepts and is supported by the case study "Invoice Management System" (IMS). The IMS requirements are outlined in brief in the following text, with more extensive details available in [6, 19]:

> *«The Invoice Management System (IMS) allows users to manage their customers, products, and invoices effectively. The IMS allows each user to have their account and be assigned to a specific user role, such as operator, manager, or customer. For each customer, the system shall maintain the following information: name, fiscal id, logo image, address, bank details, and additional information such as personal contact information. Additionally, each product within the system must be associated with a VAT category and have the respective current VAT value. The IMS utilises computed fields to enhance user productivity further, allowing users to focus on only the most essential information. [...]»*

```
2   Package ims
3
4   DataEntity e_VAT "VAT Category" : Reference [
5   attribute VATCode "VAT code" : String [constraints (PrimaryKey NotNull Unique)]
6   attribute VATName "VAT name": String
7   attribute VATValue "VAT value": String
8   ]
9
10  DataEntity e_Product "Product" : Master [
11  attribute ID "Product ID" : Integer [constraints (PrimaryKey NotNull Unique)]
12  attribute productName "Product Name" : String(50)
13  attribute vatCode "VAT code" : Integer [ constraints (NotNull ForeignKey(e_VAT)) ]
14  attribute valueWithoutVAT "Value Without VAT" : Decimal(16.2) [constraints (NotNull)]
15  attribute valueWithVAT "Value With VAT" : Decimal(16.2) [constraints (NotNull)]
16  attribute VATValue "VAT value" : Decimal [ expression arithmetic (e_VAT.VATValue) ]
17  ]
18
19  DataEntity e_Customer "Customer" : Master [
20  attribute ID "Customer ID" : Integer [constraints (PrimaryKey NotNull Unique)]
21  attribute customerName "Name" : String(50)
22  attribute fiscalID "Fiscal ID" : String(9)
23  attribute logoImage "Logo image" : Image
24  attribute address "Address" : String(100)
25  attribute IBAN "IBAN": String(34)
26  attribute SWIFT "SWIFT" : String(8)
27  ]
```

Fig. 1. Partial IMS specification in IT-Lingo ASL.

4.1. Data Entities

In ITLingo ASL, the construct known as DataEntity is used to define domain concepts or information entities like objects, persons, or business transactions. A data entity is a structural entity with attributes, foreign keys, and other specified data constraints [9, 18].

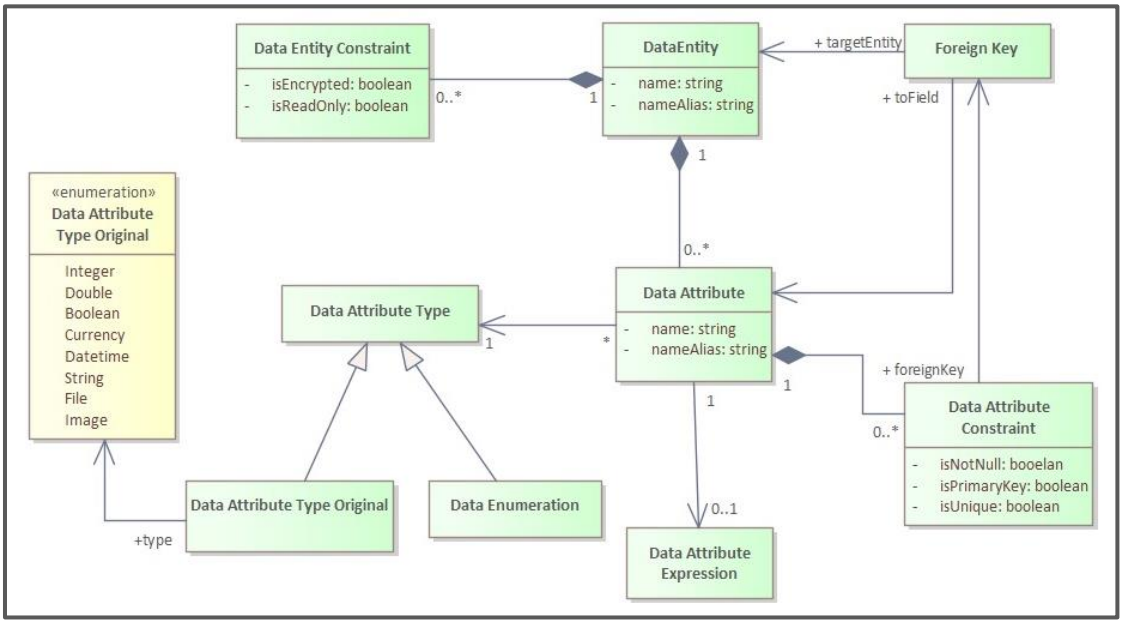

Fig. 2. Partial ASL metamodel with data entities related constructs (UML notation).

The ASL metamodel for data specification includes the concepts of DataEntity, DataAttribute, and Constraints, as shown in Figure 2. A DataEntity represents a data element and is defined by a set of DataEntityConstraints and DataAttributes. DataAttributes, in turn, represent specific data values of a given type (such as Integer or String) and are characterised by their own set of DataAttributeConstraints. These constraints can be used to specify that a particular DataAttribute is a primary key of a DataEntity, to reference other DataAttributes as foreign keys, and so on.

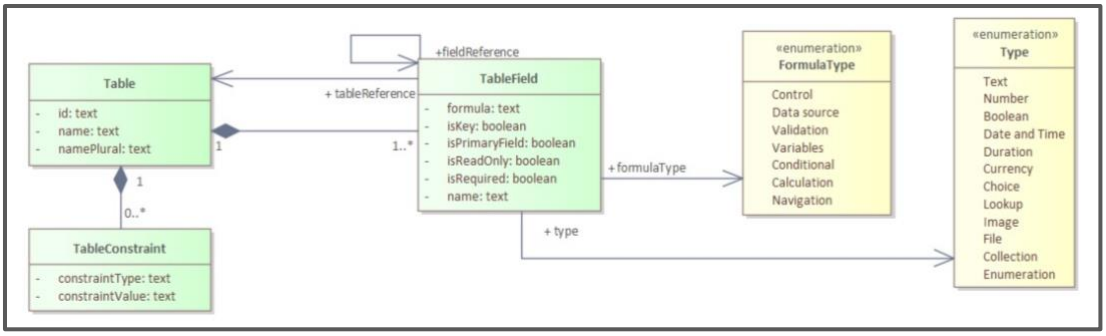

Fig. 3. Partial PowerApps metamodel with data entities related constructs (UML notation).

In contrast, Figure 3 displays a fragment of PowerApps metamodel that centres on Tables and associated constructs. In PowerApps, a Table is a data collection organised in rows and columns. It is a data source that stores data in a structured way and can be used to create forms, reports, and other visualisations. For example, following our case study, we might have a Table "Invoice" that contains fields like InvoiceID, CreationDate, and IsApproved. TableFields are the individual pieces of data that make up a table. Each field has a data type, such as text, number, date, or time, and can also have additional properties like a name. TableConstraints enforce rules or restrictions on data entered into a table, such as defining primary keys, ensuring field uniqueness, and enforcing minimum/maximum values. They are used to guarantee the integrity of the data and maintain consistency in the application.

Upon analysing the two metamodels, it becomes clear that they share common concepts. However, certain aspects, such as a field's data type, cannot be directly translated and require

additional mapping between the two representations. Furthermore, there is an opportunity to extend ASL based on a concept that exists in PowerApps and does not exist in ASL. In PowerApps, a table field can be designated as a primary field to indicate its suitability for representing the record on the user interface. While a primary key is helpful for uniquely identifying a record during internal operations, it may not provide sufficient information to the application users when displayed on the screen. Thus, PowerApps uses fields designated as primary keys, such as "Name", as potential candidates to represent the table "Customer." However, this type of specification in ASL is not currently available, which represents the opportunity to create an extension to the ASL language for it to grow as an independent language.

4.2. UI Elements

For the specification of UI elements, ASL follows the IFML terminology, in which the UI structure is defined with UI containers, components, and parts. The rules for expressing such elements are based on the IFML (Interaction Flow Modeling Language) [19].

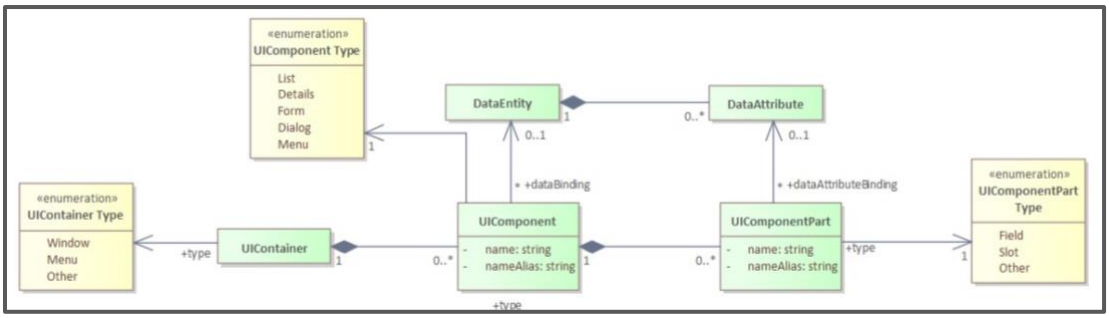

Fig. 4. Partial ASL metamodel UI-related constructs (UML notation).

UI components supported by ASL include lists, details, forms, dialogues, and menus [21]. A UIContainer collects a group of UIComponents, and each UIComponent is connected to a specific DataEntity and comprises UIComponentParts. UIComponentParts are linked to a particular DataAttribute and can be classified as a field, which is visible to the user, triggers events, and can accept parameter values, or as a slot, which is a value placeholder that remains hidden from the user [8]. A UIComponent can specify diverse UI elements, such as Forms, which PowerApps directly offer.

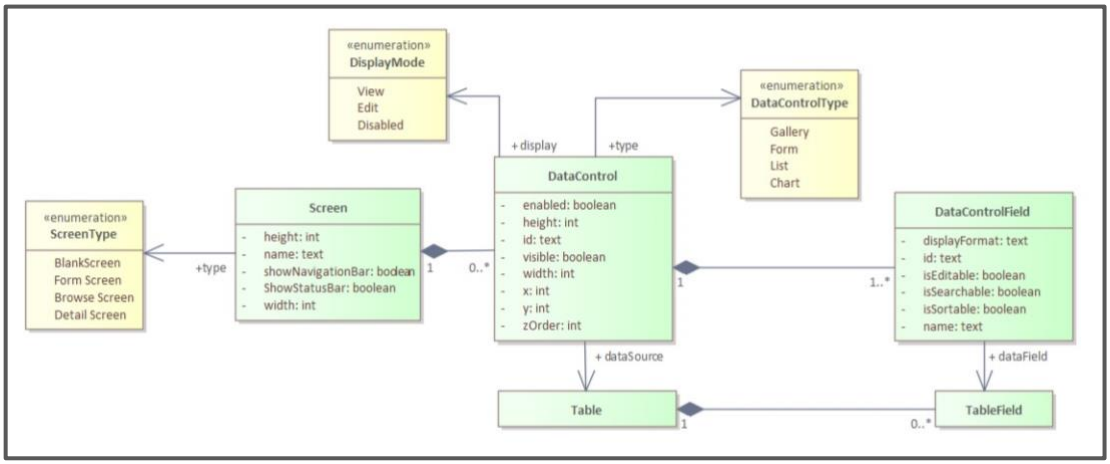

Fig. 5. Partial PowerApps metamodel UI-related constructs (UML notation).

In PowerApps, a Screen represents a visual interface that a user interacts with. It can contain various DataControls and can be used to display or edit data from a Table. A DataControl is a component that allows you to connect to a Table and display and manipulate data from that Table. Examples of DataControls in PowerApps include galleries, forms, and lists. On the other hand, a DataControlField is a specific field or column in a Table that is displayed in a DataControl. For example, if you have a Table “Customers”, a DataControlField could be the customer's name, address, or email. DataControlTypes in PowerApps refer to the different types of DataControls available, each with its own features and capabilities. PowerApps

DataControls, like forms and lists, can be directly mapped to ASL, allowing the key concepts of a PowerApps DataControl to be represented using ASL's UIComponent without any additional work since the DataControl and its fields can be defined as a UIComponent and UIComponentPart respectively. Screens can also be directly mapped into UIContainers.

4.3. Actors and use cases

In ASL, a series of interactions between an actor (or actors) and the system under study that provide the actor with value is known as a use case [9, 18].

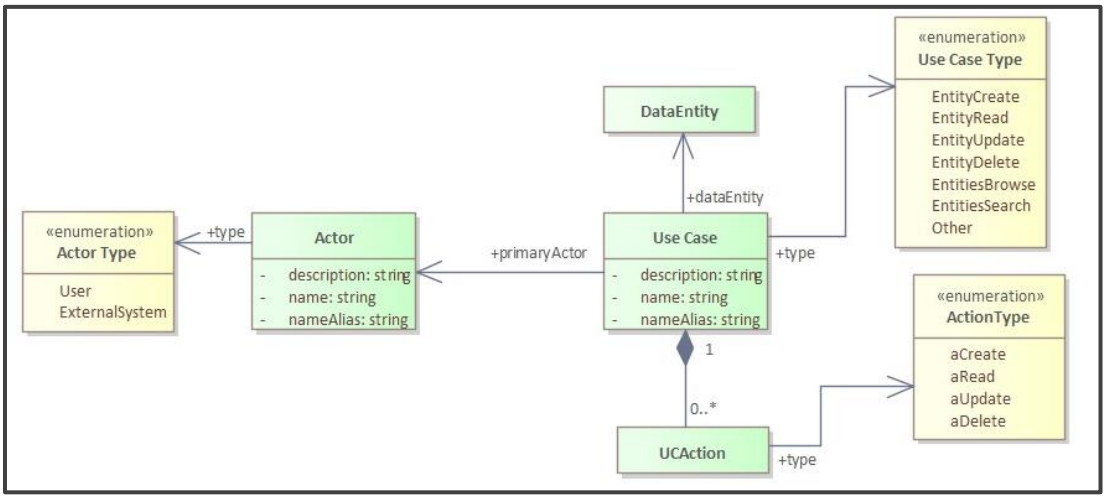

Fig. 6. Partial ASL metamodel focused on actors and use cases (UML notation).

Actors represent the entities interacting with the system: user roles or external systems. Use cases represent a sequence of actions the system's actors perform, such as CRUD operations and other specific actions (e.g., approve, export) [19, 27]. A UseCase comprises a collection of UCActions that revolve around a specific DataEntity and are driven by a primary Actor. This primary Actor can either be an end user or an external system. UseCases can be categorised into various UseCaseTypes, while each UCAction is classified as an ActionType [8]. Although these concepts cannot be directly mapped to PowerApps elements, Actors can be represented as security roles. Security roles define how users can access different records since they contain record-level and task-based privileges [33]. To define and manage security roles in PowerApps, we can create the roles and assign them to individual users or groups. We can also configure each role's permissions and access levels, including which Tables and TableFields can be accessed and which actions can be performed.

4.4. Discussion

We introduced essential concepts for specifying software business applications and explored how ASL and PowerApps languages approach them. We found that PowerApps has equivalent constructs to most of the concepts that can be expressed using ASL, which indicates that the model-to-model transformation is achievable. It is important to note that the PowerApps language is more comprehensive, providing concepts not intended to be modelled by a platform-agnostic specification language like ASL, so it is relatively easy to find ASL equivalent concepts in PowerApps, but the opposite is not.

5. ASL based transformations

This section overviews the proposed solution combining ASL specifications with PowerApps for rapidly developing software business applications. The proposed solution follows the previous work by Galhardo and Silva [8] and is based on a series of ASL-based transformations on the ASL generator and the ASL importer.

5.1. Overview of the proposed solution

Fig. 6 shows the essential tasks of the proposed approach, tackling the required transformations: ASL2ASL, ASL2MSPA, and MSPA2ASL, further explored in Section 5.3. It also illustrates the creation of an MSPA project using the generated files.

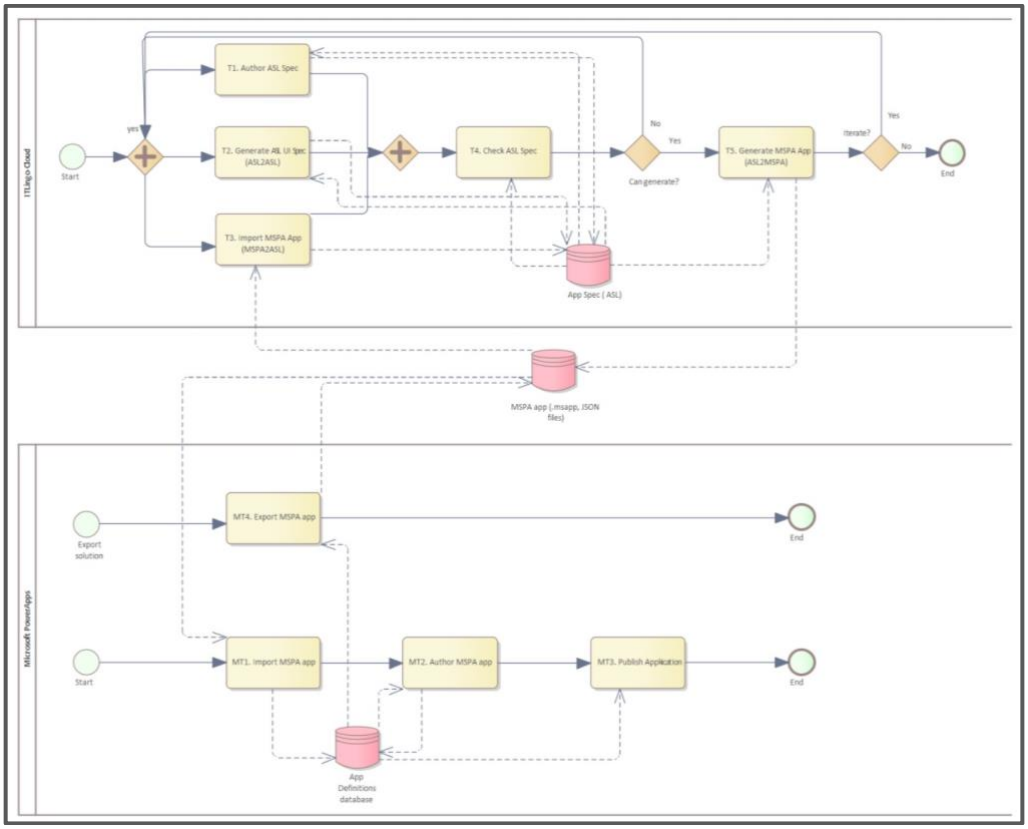

Fig. 6 Proposed solution: Combining ASL with the Microsoft PowerApps platform (BPMN notation) [8].

5.2. Generic ASL Generator and Generic ASL Importer

The ASL generator is a module that performs model-to-model and model-to-code transformations on ASL specifications as input to jumpstart the development of software business applications that follow the original requirements specification [8]. The generator is composed of (1) a platform-independent component that handles the initial loading and interpretation of ASL specifications, (2) a component that specifies the methods each concrete extension must complete, and (3) the set of extensions to the generator based on a specific target platform. The ASL importer is a module similar to the ASL generator in terms of architecture. The difference is that this module performs the opposite operation of the generator: the importer takes specifications of software business applications written in the proprietary language of their low-code platform as input and converts them to correspondent ASL specifications. The importer is composed of (1) a platform-independent component that implements generic methods essential to import and generate ASL specifications, (2) a component that specifies the methods each concrete extension must complete, and (3) the set of extensions to the importer, correspondent to a specific target platform.

5.3. ASL2ASL, ASL2MSPA, and MSPA2ASL Transformations

Several transformations must be performed to make interoperability between software business application specifications in ASL and Microsoft PowerApps possible.

Following both the work of Gamito and Silva [9] and Galhardo and Silva [8], the ASL2ASL transformation will take advantage of the automatic generation of UI specifications. This transformation allows the generation of complex ASL specifications regarding UI, based on the existing data entities and use cases. The ASL2MSPA transformation has the goal of generating a PowerApps model that is equivalent to the ASL specification: a task supported by the mapping of concepts previously explored in Section 4; Finally, the MSPA2ASL transformation is a process that performs the reverse operation to the ASL2MSPA, meaning that it takes specifications written in the proprietary language as input and converts them to correspondent ASL specifications.

6. Conclusion

This paper presents the background and motivation for this research which is a work in progress, addressing the rapid production of enterprise applications in a low-code environment. It discusses an approach that combines enterprise application specifications written in ITLingo ASL with the capabilities of Microsoft PowerApps. This approach intends to address the following issues: how to increase productivity in developing software business applications, minimising the limitations of interoperability that low-code development platforms often present; and discuss some existing solutions, namely those most related to the model-to-model transformations and model-driven approaches. Most specifically, we intend to continue the work of Galhardo and Silva [8], elevating their approach to Microsoft PowerApps [24]. A series of ASL-based transformations support this approach, now applied to the latter: ASL2ASL, ASL2MSPA, and MSPA2ASL, providing round-trip interoperability between ASL and PowerApps. It is important to note that PowerApps publishes the model at runtime, so it is easier to publish applications in this LCDP than Genio [27] since the latter performs code generation, which is more costly. Another factor that is also relevant to note is that with the possibility of integrating ASL in a cloud environment - ITLingo Cloud - it will be more practical to conduct experiments and usability tests with users. One limitation is that we do not present real-world examples because the transformations have not yet been developed.

This approach empowers developers to create precise specifications in a language not tied to a particular platform while still utilising the wide range of features of low-code platforms. As the tool expands its support for diverse low-code platforms, developers can validate their code generation capabilities without needing to gain expertise in each platform beforehand. Moreover, because ASL specifications are platform-agnostic, they encourage interoperability among different low-code applications.

Acknowledgements. Research partially funded by FCT UIDB/50021/2020.

References

1. Bézivin, Jean & Bruneliere, Hugo & Jouault, Frédéric & Kurtev, Ivan: Model engineering support for tool interoperability. (2005)
2. Bock, A.C., Frank, U.: Low-Code Platform. Bus Inf Syst Eng. 63 (6), 733–740 (2021)
3. Bragança, A., Azevedo, I., Bettencourt, N., Morais, C., Teixeira, D., Caetano, D.: Towards supporting SPL engineering in low-code platforms using a DSL approach. In: 20th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences. pp. 16–28. ACM (2021)
4. Cabot, J.: Positioning of the low-code movement within the field of model-driven engineering. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings. pp. 1–3. ACM, (2020)
5. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns, Addison-Wesley. (2001)
6. Da Silva, A.R., Savić, D.: Linguistic Patterns and Linguistic Styles for Requirements Specification: Focus on Data Entities. Applied Sciences. 11 (9), 4119 (2021)
7. Edona Elshan, Ernestine Dickhaut, Philipp Ebel: An Investigation of Why Low Code Platforms Provide Answers and New Challenges. In: Proc. of HICSS (2023)
8. Galhardo, P., da Silva, A.R.: Combining Rigorous Requirements Specifications with Low-Code Platforms to Rapid Development Software Business Applications. Applied Sciences. 12 (19), 9556 (2022)
9. Gamito, I., Da Silva, A.R.: From Rigorous Requirements and User Interfaces Specifications into Software Business Applications. In: Proc. of QUATIC, pp. 459–473. Springer, (2020)
10. Gurcan, F., Taentzer, G.: Using Microsoft PowerApps, Mendix and OutSystems in Two Development Scenarios: An Experience Report. In: ACM/IEEE MODELS-C. pp. 67–72. IEEE, (2021)

11. Kosar, T., Oliveira, N., Mernik, M., Pereira, V., Crepinsek, M., Da, C., Henriques, R.: Comparing general-purpose and domain-specific languages: An empirical study. ComSIS. 7 (2), 247–264 (2010)
12. Kurtev, I., Bézivin, J., Jouault, F., Valduriez, P.: Model-based DSL frameworks. In: Companion to the 21st ACM. pp. 602–616. ACM OOPSLA, (2006)
13. Prinz, N., Rentrop, C., Huber, M.: Low-Code Development Platforms-A Literature Review. AMCIS. (2021)
14. Da Silva, A.R.: Model-driven engineering: A survey supported by the unified conceptual model. Computer Languages, Systems & Structures. 43 139–155 (2015)
15. Sahay, A., Indamutsa, A., Di Ruscio, D., Pierantonio, A.: Supporting the understanding and comparison of low-code development platforms. In: Euromicro (SEAA). pp. 171–178. IEEE (2020)
16. dos Santos Soares, M., Vrancken, J.L.: Model-Driven User Requirements Specification using SysML., Journal of Software (2008)
17. Shah, T., V Patel, S.: A Review of Requirement Engineering Issues and Challenges in Various Software Development Methods. IJCA. 99 (15), 36–45 (2014)
18. da Silva, A.R.: ITLingo Research Initiative in 2022, http://arxiv.org/abs/2206.14553, (2022)
19. da Silva, A.R.: Linguistic Patterns, Styles, and Guidelines for Writing Requirements Specifications: Focus on Use Cases and Scenarios. IEEE Access. 9 143506–143530 (2021)
20. da Silva, A.R.: Rigorous Specification of Use Cases with the RSL Language. International Conference on Information Systems Development, (2019)
21. Talesra, K., G. S., N.: Low-Code Platform for Application Development. International Journal of Applied Engineering Research. 16 (5), 346 (2021)
22. Van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: an annotated bibliography. SIGPLAN Not. 35 (6), 26–36 (2000)
23. Voelter, M., Visser, E.: Product Line Engineering Using Domain-Specific Languages. In: 2011 15th International Software Product Line Conference. pp. 70–79. IEEE, Munich, Germany (2011)
24. Build applications faster-with fewer resources, *Business Apps | Microsoft Power*, https://powerapps.microsoft.com/en-gb/, Accessed: April 14, 2023
25. Take care of what IS important. Automate the rest., *Power Automate | Microsoft Power Platform*, https://powerautomate.microsoft.com/pt-pt/, Accessed: April 10, 2023
26. Mendix Evaluation guide, https://www.mendix.com/evaluation-guide/, Accessed: April 20, 2023, (2022)
27. Genio: Platform xtreme low-code, Quidgest,https://quidgest.com/quidgest/plataforma-genio/, Accessed: April 16, 2023, (2022)
28. Interaction Flow Modeling Language, https://www.omg.org/spec/IFML/1.0, Accessed: April 04, 2023
29. Microsoft | Power BI, *Microsoft | Power BI*, https://powerbi.microsoft.com , Accessed: April 19, 2023
30. Microsoft | PowerPlatform, https://powerplatform.microsoft.com, Accessed: April 11, 2023
31. What is microsoft dataverse?- power apps, Power Apps | Microsoft Learn, https://learn.microsoft.com/pt-pt/power-apps/maker/data-platform/data- platform-intro, Accessed: April 05, 2023
32. Outsystems Evaluation Guide, *OutSystems*, https://www.outsystems.com/evaluation-guide/, Accessed: April 15, 2023
33. Security roles and Privileges - Power Platform, https://learn.microsoft.com/en-us/power-platform/admin/security-roles- privileges, Accessed: April 17, 2023
34. Start building apps - power apps, *Start building apps - Power Apps | Microsoft Learn*, https://learn.microsoft.com/en-us/power-apps/maker/, Accessed: April 11, 2023