CNL2C: An Editor, Syntax Checker and Code Generator for CNUIML using Xtext and Xtend

H.Bahri[1], H.Motameni[2*], B.Barzegar[3]

Abstract–End-User Development (EUD) is a dynamic research area in computer science, focusing on empowering end-users to create and modify software through various approaches, constantly evolving with new methods and tools. To enhance end-user participation, research suggests developing user-friendly tools for end-users to design the UI, with the final source code derived from analyzing and automatically transforming this UI. Controlled natural language programming uses a limited version of a natural language for coding. This approach enhances programming accessibility by enabling end-users to code in a familiar language, maintaining the necessary precision and clarity. This research study the development of the CNUIML language and generation of an editor for it using Xtext. It delves into syntax error checking and target code generation using Xtext and Xtend. The CNUIML language is used to describe web application interfaces, focusing on system requirements and end-user concerns. Web applications consist of interconnected pages and forms, forming a tree of objects. Each application is a set of forms with specific value types and domains. In order to evaluate the usability of the designed tool, we have used a case study. This case study demonstrates the process of creating CNUIML models and generating the associated HTML codes using CNL2C.

Keywords: User interface description language, Model-driven user interface modeling, Xtext, Xtend, Automatic code generation

1. Introduction

Software development is a costly, complex process with the risk of not achieving the original goals, or with unusually high resource or time requirements. For this reason, many efforts have been made in research and development to reduce waste and increase the success rate of software projects[1].

According to one of the principles of Lean Thinking, the shorter the time span between the initial understanding of the problem and the presentation of a partial or final solution, the less the waste of resources and the higher the project success[2].In this regard, the new methods of software development are based on the agility of the development process and rely on a set of basic principles and guidelines.

One of the approaches considered in the research literature to address the above challenge is to leverage the skills of experienced and capable users who are able to actively participate in the development process from the earliest stages and directly contribute to the development of the final result[3].End-User Development (EUD) is an emerging paradigm in software development that focuses on enabling end-users, who are not professional software developers, to create, modify, or extend software artifacts[4]. This can increase the flexibility and usability of software systems, and can also help to bridge the gap between end-users' high domain knowledge and their limited programming expertise[5].

End-User Development (EUD) is an active research topic within the field of computer science and human-computer interaction. Various EUD approaches exist, including natural language programming, spreadsheets, scripting languages (particularly in an office suite or art application), visual programming, trigger-action programming, and programming by example[6]. These are just a few examples of the many EUD approaches represented in research and literature. The field is constantly evolving as new methods and tools are developed to empower end-users to create and modify software artifacts.

Model-driven software development (MDSE) is a software development approach that focuses on the use of

1 Department of Computer Engineering, Babol Branch, Islamic Azad University, Babol, Iran. Email:h.bahri@iausari.ac.ir

2* Corresponding Author :Department of Computer Engineering, Sari Branch, Islamic Azad University, Sari, Iran.

Email:h_motameni@yahoo.com

3Department of Computer Engineering, Babol Branch, Islamic Azad University, Babol, Iran. Email:barzegar.behnam@yahoo.com

Received: 2024.04.05; Accepted: 2024.06.09

models as the primary artifacts of the development process. In MDSE, models are used to represent different aspects of the software system, such as its structure, behavior, and requirements. These models are then used to automatically generate code and other artifacts through model transformations.

While MDSE can help to improve the quality of software systems by providing a high-level, abstract view of the system and enabling automated code generation, it is not typically considered an approach to End-User Development (EUD).EUD focuses on enabling end-users, who are not professional software developers, to create, modify, or extend software artifacts. This is achieved through a set of methods, techniques, and tools that are designed to be accessible and usable by non-professional developers.

In contrast, MDSE typically requires a high level of expertise in modeling languages and model transformation techniques, which may not be accessible to end-users. However, it is possible that some aspects of MDSE could be incorporated into EUD approaches to provide additional support for end-users in creating and modifying software artifacts.

Although standard transformation methods have been proposed in recent approaches to transform the initial and intermediate models into the final code, this still limits the end-user participation in the development process, because one of the main challenges is the difficulty of creating conceptual models by the end-user [7]. Most end users lack the necessary skills to develop such models.

Controlled natural language programming has been the subject of research within the field of EUD, and various tools and techniques have been developed to support this approach. Controlled natural language programming is a type of natural language programming that uses a restricted subset of a natural language to express program instructions. This approach aims to make programming more accessible to end-users by allowing them to write programs using a language that is familiar to them, while still providing the precision and unambiguity required for programming.

A controlled natural modeling language for form-based software development is presented in [8] by the authors.To develop this modeling language, the MDA approach has been used.

Model Driven Architecture (MDA) is a software design approach for the development of software systems that falls under the umbrella of Model-Driven Software Engineering (MDSE)[9].MDA provides a set of guidelines for the structuring of specifications, which are expressed as models. It supports model-driven engineering of software systems and is a kind of domain engineering. MDA was launched by the Object Management Group (OMG) in 2001. In MDA, models are created at varying levels of abstraction, from a conceptual view down to the smallest implementation detail. OMG literature speaks of three such levels of abstraction, or architectural viewpoints: the Computation-independent Model (CIM), the Platform-independent model (PIM), and the Platform-specific model (PSM). The CIM describes a system conceptually, the PIM describes the computational aspects of a system without reference to the technologies that may be used to implement it, and the PSM provides the technical details necessary to implement the system.

Today's user interfaces (UIs) are complex software components, which play an essential role in the usability of an application. A user interface model is a representation of how the end user interacts with a computer program or another device and also how the system responds. In many software projects, a lot of time is spent on user interface development (UI). Research in the early 1990s found that about 48% of source code and 50% of application development time was spent implementing user interfaces [10]. These figures are still acceptable today, especially given the increasing demand for graphical and web-based user interfaces [11]. Therefore, one of the ideas pursued in the research literature to increase end-user participation in the development process is the development of an easy-to-use tool that allows the end-user to design the user interface of the final product. Based on this method, the final source code is obtained from the analysis and automatic transformation of this user interface[12].

Model-based development has also influenced the development of web applications in the last decades. In [8], we investigated the development of rich Internet applications using user interface models. User interface is the critical factor for web application adoption[13].In this research we want to investigate the development of a tool for modeling the user interface of form-based web applications using EMF and Xtext Language. This tool supports controlled natural language described in [8].

The Eclipse Modeling Framework (EMF) is a modeling framework and code generation facility for building tools and other applications based on a structured data model. EMF is a powerful tool for building tools and applications based on structured data models. It provides a flexible and extensible framework that can be used to develop a wide range of applications. EMF is commonly used as a standard for UI frameworks, and support for transformations [14].

Xtext is an open-source software framework for developing programming languages and domain-specific languages (DSLs). Unlike standard parser generators, Xtext

generates not only a parser, but also a class model for the abstract syntax tree, as well as providing a fully featured, customizable Eclipse-based IDE. Xtext is being developed in the Eclipse Project as part of the Eclipse Modeling Framework Project and is licensed under the Eclipse Public License.

The Xtext tool is an open source framework for programming language development and specifically for the DSLs that can be used to define a language using a powerful grammar language. As a result, we have a complete infrastructure, including parser, linker, type checker, compiler, as well as editing support for Eclipse and any other editor that supports the language server protocol and a web browser [15].

The main contributions of this research are as follows:

- *We have developed a controlled natural modeling language based on CNUIML meta-model as a user interface modeling language using Xtext, to enable end-user describe the initial description of functional requirements.*
- *We describe how source code in HTML & CSS is generated from user interface model automatically.*

The structure of this paper is as follows, we provide an overview of the research community's efforts to create appropriate frameworks and tools for the development of the model-driven user interface design tools and its transformation into final code in Section 2. In Section 3, we present an overview of CNUIML, which is presented in [8] by authors. Then, in Section 4 we describe the CNL2C, an editor for CNUIML created by Xtext and EMF. To evaluate the usability of CNL2C, in Section 5 we implement the user interface of a sample system (case study) using the described language. In Section 6 we discuss the capabilities, shortcomings, and weaknesses of CNL2C make suggestions for improving the current solution and strategies for developing and extending the capabilities of the current research.

2. Related works

Various approaches to modeling user interfaces have been proposed in the literature. Model-based user interface development environments (MB-UIDE) approach consists of a method to generate UIs from a set of declarative and high-level models and the necessary supporting tools to assist the modeling task and/or the automatic generation of the UI[11]. Other approaches include Domain modeling, Navigation modeling and Task modeling.

MDE has been applied to user interface development to increase the level of abstraction, improve the management of complexity and evolution, and maximize productivity. Model-Driven User Interface Development (MDUID) support the efficient development of user interfaces through the use of abstract modeling and transformation to final user interfaces. Widely studied approaches include UsiXML, MARIA, and IFML[16].

User interface modeling languages can be categorized into two main types: textual and graphical. Textual modeling languages use standardized keywords accompanied by parameters or natural language terms and phrases to make computer-interpretable expressions. An example of a textual modeling language is EXPRESS, which is both a graphical and textual modeling language. Graphical modeling languages use a diagram technique with named symbols that represent concepts and lines that connect the symbols and represent relationships and various other graphical notation to represent constraints. Some examples of graphical user interface modeling languages include MARIA XML, UMLi, UsiXML, DiaMODL, Himalia, and IFML.

IFML is a powerful tool for modeling user interactions and front-end behavior in software systems. It provides a flexible and extensible framework that can be used to develop a wide range of applications. The focus of IFML is on the structure and behavior of the application as perceived by the end user. IFML was developed in 2012 and 2013 under the lead of WebRatio and was inspired by the WebML notation, as well as by a few other experiences in the Web modeling field. It was adopted as a standard by the Object Management Group (OMG) in March 2013.

Erazo-Garzón et al.[17] show that, articles on MDE applied to the UI began to be published starting in 2005. They claimed (70.59%) of article apply the models only at design time to increase the abstraction and automation of the UI artifact development process and 45.10% of studies are focused on the web. Several studies (39.22%) aligned with the Model-Driven Architecture (MDA). Almost half of the studies (45.10%) suggested a DSL tool and only 35.29% suggested model-to-text conversion. Most of the tools used by researchers to build DSLs are EMF (Eclipse Model Framework) and only a few of them have used Xtext. The approaches studied, generate the code of the target UI artifacts mostly in languages such as Java, JavaScript and HTML, and Python, C++, C\#, Objective-C, XAML, ASP.NET languages are less used. The results show that the papers that carried out a case study represent 31.37% and 76.47% represent the research papers developed in an academic environment.

A textual domain-specific language and a corresponding meta-model introduced by Mart to describe user interaction

using abstract UI patterns that can be transformed into more platform specific UI patterns during application generation[18].

Language and metamodels consist of separate parts that describes the aspects and levels of detail of the interactive elements provided to the system users during each interaction step and all the possible interactions provided by the user interface.

The primary goal of user interface description language is to capture the sets or partitions of user interface elements that are made available to system users during each interaction step. Interaction elements referenced in the language are represented as abstract patterns. The most basic abstract patterns including data entry, secure data entry, list of selectable items and event firing. The authors of the article demonstrate the grammar definition using the Xtext DSL grammar definition and suggests that the model can be transformed into both a graphical user interface and a conversational user interface, such as a voice or command line but How the AUI is transformed to the FUI model is not described.

Lachgar and Abdali, in [19] proposed a new approach to UI design for mobile applications that would be generalized to all mobile platforms and web-based components, by defining a single language, completely shared for GUI development, called Technology Neutral DSL. Their approach proposes an Android GUI meta-model to design a graphical user interface model of an Android application. Then M2M (Model to Model) and M2T (Model to Text) transformations are applied to generate GUI code targeting a specific platform. They generate their DSL with Xtext and validate its semantics using the Xtend stub automatically generated by Xtext. They have also done model-to-model and model-to-text transformation based on their DSL using Xtend.

In[20], the MDA approach is used to build a textual user interface modeling language. The architecture is structured in three main levels: At level 1, the development of mobile metamodels is addressed. This meta-model takes into account graphical user interface modeling, navigating between screens, menus, multimedia components, events, component types (date, text, email, number, etc.), the container, configuration files, and resources needed to run an application. Level 2 describes how to develop PIM-mobile. After defining the meta-model, we can create a textual model that represents the mobile app. In this step, validation for the model have implemented. In level 3 code generator was built. The code generator consists of two main blocks: The transformation of PIM-Mobile to PSM using Xtend 2 and Projection to generate the source code from the resulting PSM. The templates were developed using Xtend 2.

In[21], the authors suggest using the meta-model to represent and specify the most common GUI (Graphical User Interface) elements in mobile platforms. They define a new DSL, described through this meta-model that allow developers to design platform-independent mobile apps. A code generation engine defined and implemented to generate native code on different platforms. Metamodels are defined in UML. The proposed DSL grammar is inspired by the previous PIM meta-model and is defined in Xtext. The starting Xtext rule of the DSL is the Application rule. The screen rule (referenced from the application rule) describes a page. This rule also contains one or more element rules. The element rule delegates to a Container rule or (|) a Widget rule. A container rule describes a container element that is used to contain other elements on the screen, including other containers. Widget rules are used to describe a generic graphic element, which can be contained by a screen or a container. It can be Button, InputText, RadioButton, etc. In this article, authors focus on generating code for the Android operating system that transforms each component of the application designed using the defined DSL, into its corresponding component(s) in the Android platform. This transformation from the PIM meta-model to the Android meta-model is released with the Xtend 2 templates.

As the literature review shows, the Eclipse Modeling Framework (EMF) in combination with Xtext and Xtend is currently the state of the art for developing MDE infrastructures.

Xtext allows the creation of domain-specific modeling languages in the form of EBNF-style grammars. Based on the concrete syntax that can be specified in an Xtext grammar, a text-based editor can be generated automatically. All generated editors have syntax highlighting, auto-completion and model validation. To allow for custom extensions, the generated plugin structures allow the implementation of additional features such as editor quickfixes, validation rules, and formatters. The generated skeleton files allow an uncomplicated start for infrastructure developers. A popular solution for the realization of template-based generators with metamodel support is provided by the Xtend framework, which perfectly matches grammar-based DSL solutions created with Xtext[22].

Ceri, S et al have presented WebML, a modeling language for website design. This language is based on

XML syntax rules and graphical notation. This language includes a set of structural models to specify the website data, the composition model that assembles the website pages, the navigation model that shows the link between pages, the presentation model that specifies the layout and graphics requirements for page creation, and the personalization model that specifies the personalization features for one-to-one content delivery[23].

Bernardi et al. presented M3D (Model Driven Development with Declare), a tool that contains three metamodels representing the main components of a web application. These metamodels are provided along with a declarative language, DECLARE, for modeling business processes. This article describes the structure of a Web application in four layers: Information, Service, Presentation and Process. UML notation is used to describe the metamodel of each of the above layers. A DSL is used to describe the models of each layer. In the code generation step, the transformation of the models into the final code based on the MVC pattern based on the J2EE framework is completed using the Xpand language[24].

Sabraoui et al. have proposed an MDA-based model-driven approach to automatically generate GUIs for mobile applications. They describe the approach in four steps, including: GUI analysis and modeling using UML, transforming models to XMI files using JDOM API, model-to-model transformation using the ATL language to convert PIM models to PSM, and using the Xpand language to transform the model to text and generate the final code[25].

Table 1. Comparison of the features of Xtext, DSL Forge, Spoofax, and JetBrains MPS

	Xtext	DSL Forge	Spoofax	JetBrains MPS
Syntax Highlighting	Yes	Yes	Yes	Yes
Syntax Validation	Yes	Yes	Yes	Yes
Content Assist/Code Completion	Yes	Yes	Yes	Yes
Semantic Validation	Yes	Server-side	Yes	Yes
Code Refactoring	Yes	No	No	Yes
Error Checking	Yes	Yes	Yes	Yes
Quick Fixes	Yes	No	No	Yes
Debugging of DSLs	Yes	No	No	Yes
Language Versioning	Yes	No	No	Yes
Code Folding	Yes	Yes	No	Yes
Text Hovering	Yes	Yes	No	Yes
Brace Matching	Yes	Yes	No	No
Scoping	Yes	Yes	Yes	Yes
Generators	Yes	Yes	Yes	Yes

There are some graphical editor tools similar to Xtext that can be utilized for developing an editor for a Domain-Specific Language (DSL). DSL Forge , Spoofax , Jetbrains MPS are among these tools and also text-based formats created with Xtext can be combined with GEF, Sirius or Graphiti too.These tools can be compared in terms of different factors. Table 1 compares the most important features of these tools.

3. CNUIML (Controlled Natural User Interface Modeling Language) Overview

In [8] , authors described a controlled metalanguage that closely resembles natural language and can be used to describe user interface elements. This language includes features such as grouping data items into entities, defining relationships between entities, and automatically recognizing data types based on sample values or descriptive expressions.

CNUIML is created based on a meta-meta-model to describe the user interface of a web application. Some features of the user interface of web applications are defined by business analysts, software system designers, or programmers, and the focus in the CNUIML is on features related to the main requirements of the system and the concerns of the end user. The most web applications consist of interconnected pages and forms, and each page may contain containers or sub-containers with related data items. This structure forms a tree of objects, where each node is a form, a sub-container, or a data item representing a single value or a set of values.

The meta-model for a web application, excluding navigation, event, action, and modularization UI features, consists of four elements: Form, Sub-form (container or subspace), Data item, and Domain range and type of values. Each web application is a set of forms containing data items with specific value types and domains. Forms can have zero or more associated sub-forms, with a one-to-many relationship between forms and sub-forms.

The range and type of data values can be specified using a limited or unlimited set of specific data types or by referring to data items in other forms or lists of homogeneous data items. Acceptable data types include integers, decimals, dates, times, letters, character strings, logical values, emails, and binary arrays like images and attachments.

The CNUIML meta-model is described using the Xtext

method. Figure1 shows part of the grammar of the language.

```
Model:
    cNUIML+=Project;

Project returns Project:
    {Project} 'project' name=IDENTIFIRE (forms+=Form_definition)*;


Form_definition returns Form:
    {Form} 'Form' 'of' name=Label 'as'
        (statements+=Statement)+
        (subform+=Subform_definition)*
        'End' 'Of' 'Form';

Statement:
    {Statement} name=Label '::' valuesExpression=Expression;

Expression:
    Single_value
    | {Setofvalues} (( svalues+=Single_value ',' )+
        svalues+=Single_value non_enumerated?=("...")?)
    | {Rangeofvalues} (rvalues+=RANGE_OF_VALUES (',' rvalues+=RANGE_OF_VALUES)*)
    | {Referencetovalues} ('a' | 'an') label=Label 'in' form=[Form|Label] 'List';
```

Fig. 1.A section of the grammar of CNUIML

Each application is viewed as a project with a unique identifier or name, encompassing a collection of forms for data display, input, and modification. This collection of forms is a series of statements that outline the forms and their associated sub-forms, concluding with "End of Form". The project ID commences with a legitimate alphabetical character and may comprise both letters and numbers.

The form's description starts with "Form of", followed by the form's descriptive ID, and further elaborates after "as". In order to describe the form, the descriptive expressions of the data items are provided first, followed by the placement of the sub-forms' description. The descriptive ID of each statement's form consists of valid letters, with spaces permitted in between them.

To describe data items, two attributes are necessary: the label or name of the data item and a sample of its valid values that define the type and value range. The "::" symbol is utilized to distinguish these two parts.

The valid value range of a data item can encompass one or more (a set) of distinct values. These individual values could be either limited or unlimited. For instance, "Man" represents a single value, "Man, Woman" signifies a finite set of values, and "Bachelor, MA, Ph.D.,..." denotes an infinite set of values. Permitted values include date, time, numbers, literal strings, email, logical values, and binary values. Binary values pertain to images or other attachments that can be uploaded in binary format.

To illustrate the value range of an expression, we employ one or more value ranges depicted as "start..end", where "start" signifies the minimum value and "end" represents the maximum value of the range. Date, time, and numbers are acceptable values for the start and end of the range.

In certain instances, the valid value range of a data item is confined to the set of values present in other project entities. Referential expressions are utilized to portray these types of ranges, which refer to the name or label of a data item from another project form.

The definition of sub-forms mirrors that of the form and includes the title of the sub-form and the title of the referenced form. The sub-form maintains a one-to-many relationship with the main form. The titles of the data elements of the sub-form can be a subset of the titles of the data items of the referenced form or all of them. The same descriptive expression is employed in the description of the data elements of the sub-forms as in the description of the forms.

4. CNL2C (Controlled Natural Language to Code) Tool

CNL2C is a full feature editor for CNUIML developed using Xtext. In this section, we describe the features of this tool and how to develop it. To generate a full-featured editor for a Domain Specific Language (DSL) using Xtext, we should implement features such as syntax highlighting,content assist,validation and quick fixes. These features are essential for creating a user-friendly and efficient editor for a DSL. Xtext provides a comprehensive set of tools and extensions to help developer implement these features and more.

To generate the editor, we should follow these steps: define the DSL, generate the language infrastructure, create the editor,customize the editor features and generate the editor. As mentioned in Section 3, the CNUIML domain-specific language is described using the Xtext language. The generator will then produce a parser, an AST-meta model (which is implemented in EMF), and a comprehensive Eclipse Text Editor based on our definition. We used Xtext to construct the language framework, which encompasses the parser, compiler, and all other essential elements. This can be accomplished by executing the Xtext generator. Xtext provides a default editor with syntax highlighting, content assist, and other basic features. Figure 2 shows a view of the Eclipse environment containing the DSL definition and the infrastructure created by the code generator.

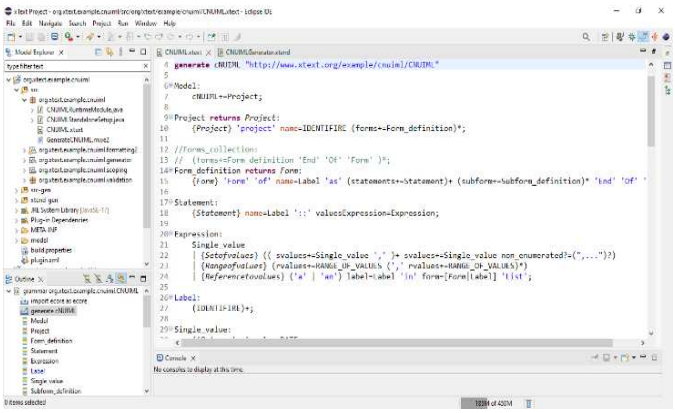

Fig. 2.A view of the Eclipse environment containing the DSL definition

Stub codes are automatically generated for formatting text, validating expressions and generating the final code,after running the project in the Eclipse environment. Stub codes can be generated using both Java and Xtend. Figure 3 shows how these snippets are used to add new lines to form definitions. The setNewLines method ensures that at least and preferably one and at most two new lines are added at the end of each form definition.

```
def dispatch void format(Project project,
    extension IFormattableDocument document
) {
    for (form : project.forms) {
        form.append[setNewLines(1, 1, 2)]
        form.format
    }
}
```

Fig. 3. Form formatter stub code

The code editor's built-in features are used to carry out numerous validations, which are governed by the syntax rules defined in the language. However, certain validations based on syntax rules can be extremely challenging or even unfeasible due to the limitations of parser evaluation.In the CNUIML language, the parser is unable to validate whether the lower and upper bounds in the RangeValues data type definition are of the same type.Such validations are checked through the use of stub codes.Figure 4, illustrates the data type validation stub code for the upper and lower boundaries of a Range.

```
@Check
public void checkRangeBounds(Ranges range)
{
    if(range.getLowerBound().getClass() !=
            range.getUpperBound().getClass())
    {
        error("Type of upper bound must be the same as lower bound : "
    + range.getLowerBound().getClass().getName()
                ,CNUIMLPackage.Literals.RANGES__UPPER_BOUND
                ,TYPE_MISMATCH);
    }
}
```

Fig. 4. Data type validation stub code for the upper and lower boundaries of a Range

The code shows that a syntax error will be displayed if the Range's upper and lower boundsare not of the same data type. Figure 5 shows the process of validating the data types of values within a set. All values in a set must be of the same data type. If a value of a different data type is detected within the set, a syntax error will be generated. The code shows the method of comparing the data type of the first value with the data types of all subsequent values in the set. The moment a mismatch is detected, a syntax error is generated.

```
@Check
public void checkSetMembersType(Setofvalues setOfVales)
{
    Single_value first = setOfVales.getSvalues().get(0);
    for ( Single_value  sval : setOfVales.getSvalues() ) {
        if ( sval.getClass() != first.getClass()  )
            error("Type of all values must be : "
                    + first.getClass().getName()
                    ,CNUIMLPackage.Literals.SETOFVALUES__SVALUES
                    ,TYPE_MISMATCH);
    }
}
```

Fig. 5. Validating the data types of values within a set

When implementing the semantics of a DSL, we focus on defining how the language behaves that includes type systems, reduction rules, interpreters, transformation rules. The process of transforming code in a Domain-Specific Language (DSL) into the source code in the target language is an essential aspect of implementing the language's semantics. This step involves generating code (e.g., HTML, CSS, Python) from DSL programs. The generated code should adhere to the semantics defined by the DSL. During code generation, DSL constructs (such as DSL expressions or statements) is mapped to corresponding constructs in the target language.

According to MDA (Model-Driven Architecture), CNUIML is classified as a Platform Independent Model (PIM). To generate the final code, it needs to be transformed into a platform-specific model (PSM). Any PIM can be transformed into various distinct platform-specific models. In our research, we focused on the form-based web application model, which can be implemented in different ways. Table 2 illustrates the sample used by the researchers, demonstrating how CNUIML language elements map to the chosen platform-specific model elements.

Table 2. Mapping table of CNUIML language components to web form application model

CNUIML Meta Model (PIM)	Web Form Application Meta Model (PSM)
Project	Project Home Page
Form.name	Link to Web Form
Form	Web Form Home Page
Form	Web Form
Form statements	Field Groups
Statement	Field Group
Expression	Input Component
Statement.name	Label

Any platform-specific model (PSM) can be transformed into a platform-specific implementation (PSI). The platformspecific implementation depends entirely on the execution environment and the target language. Table3 shows the mapping table of CNUIML language components to HTML5 language.

Table 3.mapping table of CNUIML language components to HTML5 language

Web Form Application Meta Model (PSM)	HTML Web Page Meta Model (PSI)
Project Home Page	<html> tag
Link to Web Form	<a> tag
Web Form Home Page	<html> tag
Web Form	<form> tag
Submit Button	<input> tag
Field Groups	<table> tag
Field Group	<tr> tag
Input Component	<input> tag
Label	<label> tag

In the context of MDA (Model-Driven Architecture) or MDE (Model-Driven Engineering), M2T (Model-to-Text) transformation plays a crucial role. This process involves converting a model into a text document, adhering to a specific metamodel. The resulting text document can serve various purposes, such as source code, documentation, or configuration scripts. Typically, this transformation is achieved by defining templates that specify how to translate model elements into textual content. These templates are often expressed using a template language. Template Language is a programming language designed to process and generate structured text. Within Template Language, specific structures allow repetitive actions to be performed on all elements of the source model. The resulting text output considers the features of these elements while accounting for necessary conditions. Stub codesto generate the final code are automatically generated using Xtext. These codes can be in Java or Xtend language. Xtend employs its unique template language for code generation. Unlike typical template languages, Xtend stands out by supporting grayspace. This distinctive feature makes Xtend a potent tool for code generation within the realms of MDE (Model-Driven Engineering) or MDA (Model-Driven Architecture). The transformation rules are created based on the mapping table. Figure 6shows the stub code responsible for generating HTML5 code from a CNUIML language program.

```
override void doGenerate(Resource resource, IFileSystemAccess2 fsa,
    IGeneratorContext context
) {
    var links = ""
    for (e : resource.allContents.toIterable.filter(Form)){
        val filename = e.fullyQualifiedName.toString("/") + ".html"
        links +=
        '''
            <a href="./«filename»">«e.name»</a><br>
        '''
        fsa.generateFile(filename,
        e.compile)
        }
        val project = resource.allContents.toIterable.filter(Project).head
        fsa.generateFile(project.fullyQualifiedName.toString("/") + ".html",
        '''
            <!DOCTYPE html>
            <html>
            <body>
            <h1>«project.name»</h1>
            <p>Autogenerated by CNUIML</p>
            «links»
            </body>
            </html>
        ''')
}
```

Fig. 6.stub code responsible for generating HTML5 code from a CNUIML language program

The method iterates over all elements (e) in the resource that are of type Form. For each Form element, it constructs an HTML link. The filename is derived from the fully qualified name of the Form element, with .html appended. The links string accumulates these HTML links. The link points to the generated .html file corresponding to the Form element. For each Form element, it generates an .html file using the fsa.generateFile method.The content of the file is obtained from the e.compile expression.e.compile is responsible for transforming each Form element into HTML5 code.The snippet assumes there is a Project element in the resource. It retrieves Project element and constructs a project-level .html file. The file contains an HTML structure with a title (project.name), an autogenerated message, and the previously generated links.Other transformation methods have a similar structure.

5. Evaluation (Case Studies)

The case study approach provides a rigorous and systematic way to evaluate the usability of an artifact in DSR, taking into account the complexity and context-specific nature of design problems[26].The case study of this research is the course management system described in [8]. The purpose of the course management system is to

record the profile of each instructor, course topics and training courses. Figure 7 shows the view of home page generated by CNL2C related to course management system forms. This view contains links to other project forms.

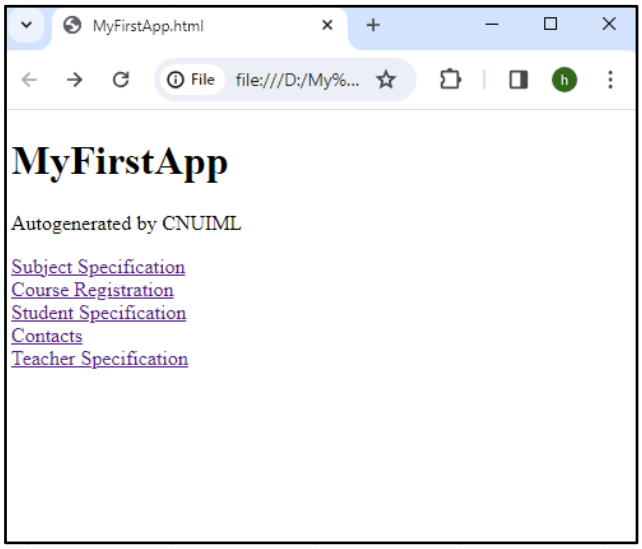

Fig. 7.view of home web page generated by CNL2C related to course management system

Figure 8 also shows one of the generated forms of the project. This view displays the course registration form, which is coded in the CNUIML language, following the specifications outlined in the program.

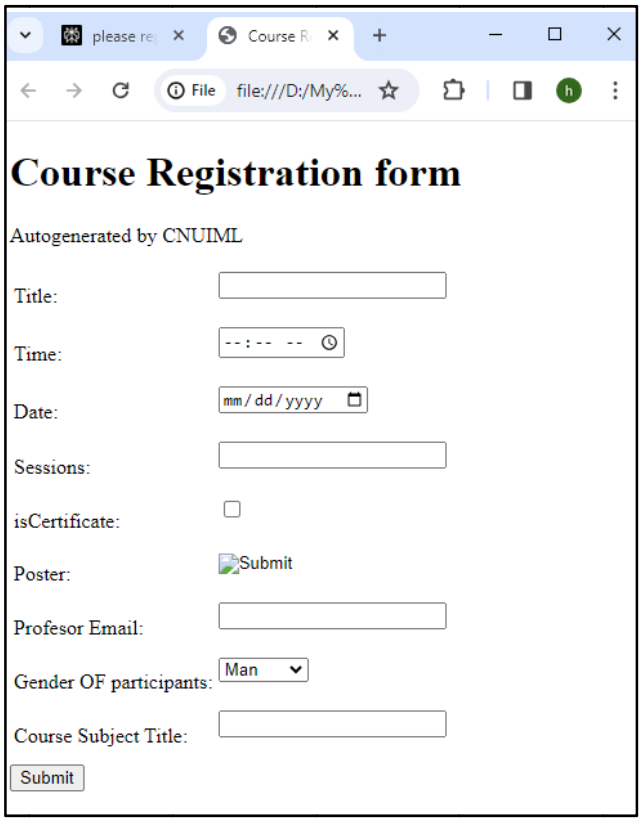

Fig. 8. Course registration form view

When the project is built, eclipse automatically generates the target code and saves it in html files. Figure 9 shows the structure of the generated files. For each form specified in the program, an HTML file has been generated, and a separate file has been created for the project entity.As can be seen, the generated code matches the expected code.

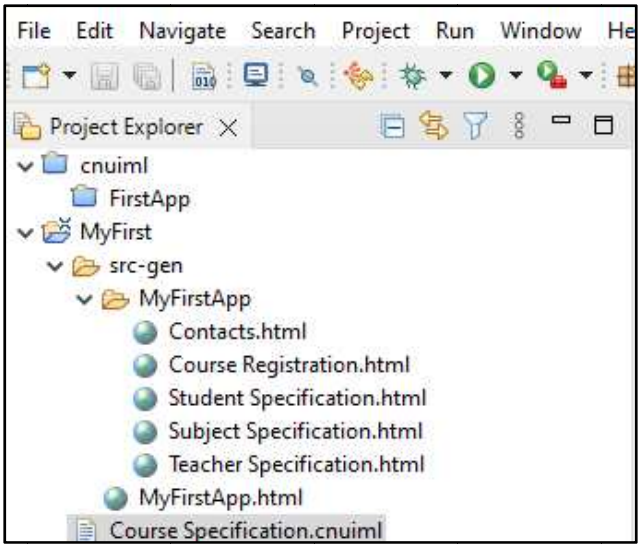

Fig. 9. Structure of the generated files

Figure 10 shows a section of the project code and the outline of the entities defined in the project. The hierarchy of all entities defined in the program is displayed as an outline. The project entity and all its forms can be seen in the figure.

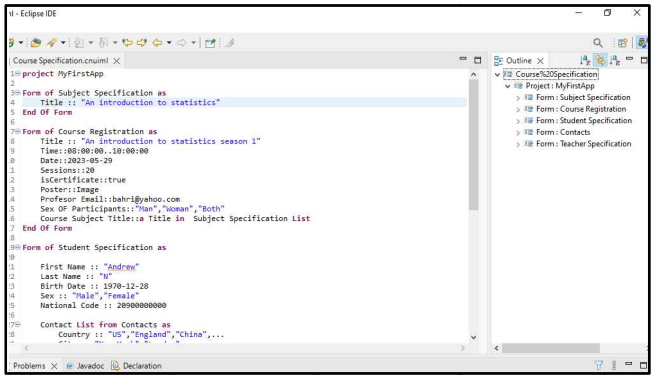

Fig. 10.A section of the project code and the outline of the entities

6. Conclusion and future works

In this study, we have examined the evolution of the CNUIML language through the use of Xtext and an editing

tool. We have also explored the detection of syntax errors and the generation of target code using Xtext and Xtend.Xtext, as a powerful framework, not only automates the creation of stub codes for validation and formatting but also manages entity scope. Furthermore, it facilitates the generation of target code. The generated code can then be customized and extended as per the requirements of the specific project or application.

The generation of transformation rules to extract equivalent models like Interaction Flow Modeling Language (IFML) or abstract models such as Unified Modeling Language (UML) class diagrams and task models is indeed a promising area of future research. Furthermore, the validation of these transformation rules is another important aspect that needs to be considered. This would ensure that the transformed models accurately represent the original models. Various validation techniques, such as simulation, formal verification, or empirical studies, could be employed for this purpose.The extraction of database code and the implementation of Create, Read, Update, and Delete (CRUD) operations, as well as the development of services and business logic layers, present a rich area for exploration and research.

References

[1] G. Kumar and P. K. Bhatia, "Impact of agile methodology on software development process," *International Journal of Computer Technology and Electronics Engineering (IJCTEE)*, vol. 2, no. 4, pp. 46-50-46–50, 2012, doi: 10.1145/2735399.2735410.

[2] B. Fitzgerald and K.-J. Stol, "Continuous software engineering: A roadmap and agenda," *Journal of Systems and Software*, vol. 123, pp. 176–189, 2017, doi: 10.1016/j.jss.2015.06.063.

[3] M. Bano and D. Zowghi, "User involvement in software development and system success: a systematic literature review," in *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, Porto de Galinhas Brazil: ACM, Apr. 2013, pp. 125–130. doi: 10.1145/2460999.2461017.

[4] H. Lieberman, F. Paternò, M. Klann, and V. Wulf, "End-User Development: An Emerging Paradigm," in *End User Development*, vol. 9, H. Lieberman, F. Paternò, and V. Wulf, Eds., in Human-Computer Interaction Series, vol. 9. , Dordrecht: Springer Netherlands, 2006, pp. 1–8. doi: 10.1007/1-4020-5386-X_1.

[5] T. Ludwig, J. Dax, V. Pipek, and V. Wulf, "A Practice-Oriented Paradigm for End-User Development," in *New Perspectives in End-User Development*, F. Paternò and V. Wulf, Eds., Cham: Springer International Publishing, 2017, pp. 23–41. doi: 10.1007/978-3-319-60291-2_2.

[6] M. Snoeck and Y. Wautelet, "Agile MERODE: a model-driven software engineering method for user-centric and value-based development," *Softw Syst Model*, vol. 21, no. 4, pp. 1469–1494, Aug. 2022, doi: 10.1007/s10270-022-01015-y.

[7] H. Soude and K. Koussonda, "A Model Driven Approach for Unifying user Interfaces Development," *IJACSA*, vol. 13, no. 7, 2022, doi: 10.14569/IJACSA.2022.01307107.

[8] H. Bahri, H. Motameni, and B. Barzegar, "CNUIML: Towards the automatic generation of enterprise-level rich Internet applications using Controlled Natural User Interface Modeling Language," *Scientia Iranica*, Dec. 2023, doi: 10.24200/sci.2023.62435.7839.

[9] D. Torre, M. Genero, Y. Labiche, and M. Elaasar, "How consistency is handled in model-driven software engineering and UML: an expert opinion survey," *Software Qual J*, vol. 31, no. 1, pp. 1–54, Mar. 2023, doi: 10.1007/s11219-022-09585-2.

[10] B. A. Myers and M. B. Rosson, "Survey on user interface programming," in *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '92*, Monterey, California, United States: ACM Press, 1992, pp. 195–202. doi: 10.1145/142750.142789.

[11] J. Ruiz, E. Serral, and M. Snoeck, "Evaluating user interface generation approaches: model-based versus model-driven development," *Softw Syst Model*, vol. 18, no. 4, pp. 2753–2776, Aug. 2019, doi: 10.1007/s10270-018-0698-x.

[12] S. Kent, "Model driven engineering," in *International conference on integrated formal methods*, Springer, 2002, pp. 286–298.

[13] K. E. Emam and A. G. Koru, "A Replicated Survey of IT Software Project Failures," *IEEE Softw.*, vol. 25, no. 5, pp. 84–90, Sep. 2008, doi: 10.1109/MS.2008.107.

[14] R. Gronback, "Eclipse Modeling Project | The Eclipse Foundation." Accessed: Aug. 08, 2023. [Online]. Available: https://eclipse.dev/modeling/emf/

[15] I. M. T. Gamito, "From Rigorous Requirements and User Interfaces Specifications into Software Business Applications: The ASL Approach," PhD Thesis, Master's thesis, Instituto Superior Técnico, 2021.

[16] E. Yigitbas, I. Jovanovikj, K. Biermeier, S. Sauer, and G. Engels, "Integrated model-driven development of self-adaptive user interfaces," *Software and Systems Modeling*, vol. 19, no. 5, pp. 1057-1081-1057–1081, 2020, doi: 10.1007/s10270-020-00777-7.

[17] L. Erazo-Garzón, S. Suquisupa, A. Bermeo, and P. Cedillo, "Model-Driven Engineering Applied to User Interfaces. A Systematic Literature Review," in *Applied Technologies*, M. Botto-Tobar, M. Zambrano Vizuete, S. Montes León, P. Torres-Carrión, and B. Durakovic, Eds., in Communications in Computer

and Information Science. Cham: Springer Nature Switzerland, 2023, pp. 575–591. doi: 10.1007/978-3-031-24985-3_42.

[18] M. Karu, "A textual domain specific language for user interface modelling," in *Emerging Trends in Computing, Informatics, Systems Sciences, and Engineering*, Springer, 2013, pp. 985–996.

[19] M. Lachgar and A. Abdali, "Generating Android graphical user interfaces using an MDA approach," in *2014 Third IEEE International Colloquium in Information Science and Technology (CIST)*, Oct. 2014, pp. 80–85. doi: 10.1109/CIST.2014.7016598.

[20] M. Lachgar and A. Abdali, "Modeling and generating native code for cross-platform mobile applications using DSL," *Intelligent Automation & Soft Computing*, vol. 23, no. 3, pp. 445–458, Jul. 2017, doi: 10.1080/10798587.2016.1239392.

[21] A. Sabraoui, A. Abouzahra, K. Afdel, and M. Machkour, "MDD Approach for Mobile Applications Based On DSL," in *2019 International Conference of Computer Science and Renewable Energies (ICCSRE)*, Jul. 2019, pp. 1–6. doi: 10.1109/ICCSRE.2019.8807572.

[22] D. Priefer, "Applying Model-Driven Engineering to Development Scenarios for Web Content Management System Extensions," 2021.

[23] S. Ceri, P. Fraternali, and A. Bongio, "Web modeling language (webML): A modeling language for designing web sites," in *Ninth International World Wide Web Conference*, Elsevier, Amsterdam, Netherlands, 2000.

[24] M. L. Bernardi, M. Cimitile, G. A. Di Lucca, and F. M. Maggi, "M3D: a tool for the model driven development of web applications," in *Proceedings of the twelfth international workshop on Web information and data management*, in WIDM '12. New York, NY, USA: Association for Computing Machinery, 2012, pp. 73–80. doi: 10.1145/2389936.2389951.

[25] A. Sabraoui, M. El Koutbi, and I. Khriss, "A MDA-based model-driven approach to generate GUI for mobile applications," *International Review on Computers and Software Journal (IRECOS)*, vol. 8, no. 3, pp. 844–852, 2013.

[26] E. Costa, A. L. Soares, and J. P. de Sousa, "Situating Case Studies Within the Design Science Research Paradigm: An Instantiation for Collaborative Networks," *Collaboration in a Hyperconnected World*, p. 531, 2016.