Model-driven development of augmented reality-based editors for domain specific languages

Iván Ruiz-Rube[1], Rubén Baena Pérez[1], José Miguel Mota[1]
Inmaculada Arnedillo-Sánchez[2]
[1] School of Engineering, University of Cádiz, Puerto Real (Cádiz), Spain
{ivan.ruiz, ruben.baena, josemiguel.mota}@uca.es
[2]School of Computer Science and Statistics, Trinity College, Dublin, Ireland
macu.arnedillo@scss.tcd.ie

Abstract. Designing visual models to describe and conceptualize objects and systems requires abstraction skills and a predisposition for visual interactions. Readily available modeling tools rely on the users' logical-mathematical and visual-spatial abilities to support modeling design. However, they fall short of mechanisms to tap into the users' bodily-kinesthetic abilities. This research presents a model-driven framework to automatically develop visual editors to work with Domain Specific Languages in tangible interaction environments. The framework is illustrated through the development of an editor of entity-relationship models supported by augmented reality. The editor usability evaluation indicates good acceptance by users as well as potential to support alternative interactions and to learn database concepts.

Keywords: Augmented Reality; Domain Specific Language; Model-Driven Development; Eclipse; Unity.

1 Introduction

In many disciplines, modeling is an essential activity to describe and conceptualize objects and systems. Models are abstractions which illustrate entities, whether physical objects or representations of reality. They may fulfil descriptive or prescriptive purposes and be applicable to different knowledge areas [27]. In computing systems, models are used to capture and analyse requirements and to design the architecture and behaviour of software components. Domain-Specific Languages (DSLs) are essential for the communication between domain experts and software engineers [22]. These languages allow experts to establish a common and shared vocabulary for either a particular domain or several ones. DSLs are usually implemented through visual or textual tools and interaction with such tools relies on the users' logical-mathematical and visual-spatial abilities.

According to Gardner's multiple intelligences theory [24], human cognitive competence is a set of abilities or intelligences, which allow people to solve problems, namely: musical, bodily-kinesthetic, logical-mathematical, linguistic, spatial, inter-personal, intrapersonal, and naturalistic. While all people have these abilities, Gardner argues that the difference between individuals lays in the degree of each ability and their combination [24]. For instance, someone with a stronger musical intelligence understands better the pitch relations and someone with a linguistic intelligence understands better the phonological features of languages. In this vein, visual and textual modelling tools often rely heavily on visual-spatial and logical-mathematical abilities, since visual editing tools involve spatial judgment, object visualization and problem solving in realistic, object-oriented environments, while text-based editors involve abstractions, reasoning and critical thinking. However, readily available software modelling tools fall short in taping into users' bodily-kinesthetic abilities.

New Tangible User Interfaces (TUIs) [28] enhanced by Augmented Reality (AR) [48] can deliver content based on the users' physical movements and provide kinesthetic interactions. For instance, AR allows developers to overlay virtual objects over a physical environment and, new multi-modal interactions, such as tactile, verbal, and gestural, offer alternative means of interacting. Thus, delivering enhanced and more realistic learning experiences [38]. AR-enhanced interfaces have been successfully implemented in learning scenarios to develop skills for the understanding of visual perception, spatial expression and orientation [3, 41]. AR-based TUIs may also be relevant for the acquisition of modelling skills. In particular, tangible AR-DSL editors could support non-technical users to learn modelling skills. Additionally, TUIs can foster creative learning [12] and, along with the AR technology, facilitate creative thinking, as during purchase decisions [29].

Against this background we explore the potential of alternative model manipulation mechanisms, beyond traditional visual or textual support tools, to provide enriched user experience. We define a framework to develop AR-model editors which relies on Model-Driven Software Engineering (MDE); a well-known approach to conduct the software life-cycle process. The framework provides a systematic approach for the design and deployment of AR-model editors, a dedicated metamodel and, a number of support tools. Following a model-driven development strategy and code generation, TUIs that work with visual models are automatically generated. Consequently, users with stronger bodily-kinesthetic abilities, who have a greater appreciation of the physical world and often use gestures and body language to communicate, may find this approach more in tune with their cognitive strengths. However, in its current state, TUIs generated with the framework does not provide complete support for these abilities, and hence they are not extensively analysed in this work.

The rest of the paper is structured as follows: First, the background to MDE and multi-modal Human-Machine Interaction (HMI) is introduced. Then, a model-driven framework to create AR editors for conventional DSLs is outlined. This is followed by the description of a user scenario of the framework designed to create AR editors of entity-relationships models and a usability study. Section 5 discusses some issues related to usability, scalability, maintainability and portability of the generated editors. Finally, conclusions and future work are provided.

2 Background and Related Works

This section provides an overview of MDE, modeling languages and reviews work relevant to the application of HMI techniques, such as AR and TUI, for learning purposes.

2.1 Model-Driven Software Engineering

MDE provides principles and techniques to formalize and represent expert knowledge in domain areas. It considers models first-class citizens in Software Engineering through the promotion of model design, development, transformation and use, to undertake the software life-cycle process [9]. A particular application of MDE is Model-Driven Development (MDD) which aims to automatically generate code by developing and transforming software models and hence, reduces complexity and development efforts [61]. The use of a modeling language is essential for domain experts and software engineers to design models that accurately capture and represent software systems. DSLs are domain-expert computer languages which allow professionals to develop and describe problem solutions: source code artifacts for their own field and discipline. However, there is a notorious lack of evaluation research for DSLs, as well as studies about the effectiveness of DSL approaches [37]. A survey of the use of modeling tools among students in software engineering courses was conducted [2]. That study revealed that major weaknesses about tools include lack of feedback, being slow to use, difficulty drawing the diagrams, not interacting well with other tools and being complex to use.

Modeling languages are characterized by two main features: abstract and concrete syntax. The prior refers to the structure of the language and the latter to its specific notation. To (semi) automate software development, the operational semantics of the language may be specified through two main strategies: code generation or model interpretation. DSLs are classified according to their notation: text-based or visual. Although, visual programming languages allow developers to construct mental representations of programs quicker than procedural [46], visual programs can be harder to read than text-based ones [26]. DSLs are also classified according to their implementation mechanism: internal (embedded into a host general purpose language) or external (independent of any other language). Moreover, the use of an editor allows users to create and manipulate models with the designed language. Textual syntax editors provide features such as: syntax coloring, templates, and code auto-completion, and visual language editors: copy & paste, drag and drop, zoom, and automatic element arrangement.

There are tools to create visual and textual external DSLs. For instance, Xtext [20], JetBrains MPS [30], MontiCore Language Workbench [51], Rascal Metaprogramming Language [35] and The Spoofax Language Workbench [58] are examples of tools to create text-based DSLs. In terms of visual DSLs, open source tools include: GMF [17], Sirius [57], EuGENia [36] and Graphiti [25]; and proprietary: MetaEdit+ [33] or Modeling SDK for Visual Studio [42], among others. Constructing graphical modelling environments is a costly and highly technical task, so that an example-based technique for the automatic generation of these environments has been developed [39]. The system synthesizes a modelling environment that mimics the syntax of the provided examples of the DSL created with specific drawing tools. However, and despite the availability of frameworks for designing DSLs and their support tools, there is still some room for improvement in relation to user experience [1].

2.2 Multi-modal Human Machine Interaction for Learning

In recent years, the field of HMI has experienced an exponential growth of technologies. For instance, tactile interaction can be achieved through sensitive surfaces and haptic devices [19]; verbal, through commands and auditory tracks [6] using voice recognition and synthesis technologies; gestural, using external gadgets that recognize gestures and movements [60]; mental, via electroencephalography devices which monitor emotions, track cognitive performance, and even control objects through the training and interpretation of mental activity patterns [55]. New learning environments, based on multimodal interaction technologies, have emerged as delivery formats available to teachers and their use in e-learning experiences have a positive effect on students' motivation and engagement [15, 49].

Similarly, taking advantage of image processing and computer vision technologies, Virtual Reality (VR) and AR are becoming increasingly prominent. While VR consists on immersing the user in a computer-generated world which replaces the real one, AR overlays virtual elements on actual views of physical environments to create a mixed reality in real time [43]. In particular, AR enhances the perception of users and improves their interactions with the real world by displaying information they cannot detect directly through their senses [4]. Immersive virtual reality environments provide users with higher levels of immersion, accuracy, and emotional empathy than a virtual reality environment that simply replicates the existing real world [32]. Other types of interfaces can enable the user to apply their intentions and actions to the virtual space directly without special devices. In these environments [7], users can interact with the virtual space with their own postures by using infrared sensors.

VR/AR technologies change the field of education, the way people learn and the way they acquire knowledge; consequently, significant transformations are needed in the development of educational experiences. For example, those technologies can be combined with haptic solutions and gamification ideas to support learning and provide diverse experiences [14]. Researchers have also highlighted the potential of AR for several learning purposes, such as explaining or evaluating topics, conducting lab experiments, playing educational games or augmenting information, among others [5]. Within the context of TUIs [21, 28], technologies that link digital data with everyday physical objects and surfaces have been used in seminal works by the MIT [40, 52]. Tangible interfaces have also been utilized to learn programming abstraction at different educational levels from school [59] to higher education [13]. Furthermore, tangible multi-touch surfaces have been explored in software development for tasks that involve software analysis [45]. Thus against this background, could tangible AR-enhanced interfaces be useful for more specific software modeling tasks?

3 A Framework for Creating Augmented Reality Editors for DSLs

This section outlines a framework that supports DSL designers to create modeling learning experiences through tangible interfaces based on AR features. The framework provides a systematic approach for the design and deployment of AR editors, a dedicated metamodel and support tools.

3.1 Method

To create AR-based domain-specific modeling tools, some steps must be undertaken:

- *Abstract syntax definition*: the first step when developing a DSL is designing its abstract syntax through concepts combination and modeling. This step is usually implemented by means of a metamodeling language [11] such as MOF, which is the standard from Object Management Group (OMG) for defining new metamodels. However, Ecore, the de-facto reference implementation of MOF, is used instead.
- *Concrete syntax design*: Once the language structure is modeled, the second step consists in designing how the language elements are going to be represented. To this end a metamodel and a support tool, to design the notation of the DSLs to be used in our AR-based editor, were developed.
- *Editor creation*: In order for the user to use the language, a dedicated AR editor is automatically generated from the abstract and concrete syntax.
- *Language semantics definition*: Finally, the meaning of the language concepts must be defined so that they can be interpreted. This step can be performed in different ways: 1. applying a model-to-model transformation to produce a new model conforming to an executable language, for example, by using the ATL [31] language and a transformation engine; 2. generating source code, for example using the Acceleo [16] template language and engine; 3. executing the input model on a virtual machine able to read and run the model.

3.2 Designing Augmented Reality Editors

The metamodel for designing AR editors for models is illustrated in Fig. 1. It is the Platform Independent Model (PIM) [34] that abstracts out the technical details of the real system and it is inspired by the flexibility of editors created with Sirius and the pragmatism of those generated with EuGENia. Its structure is defined below.

An *Editor* (root metaclass of the metamodel) is made up of a *Layer* and a *Toolset*. The layer presents the visual notation of the model elements and the toolset comprises the required tools to manipulate the elements.

A model is visually depicted by a set of *Nodes* and *Edges*, which in turn can include *Labels* and are all rendered on the real image captured by the camera of the device. Nodes can be represented by 3D *Models, Geometric Shape*s or *2D Images*. Different *Edges Styles* (solid, dashed or dotted lines) and *Label Styles* can be used for both lines and texts. Nodes also have a *containementKind* feature which allows designers to indicate how the nodes are visually contained in others. In this way, nodes can contain other nodes by using a horizontal, vertical or free embedded layout or using external links. All the virtual elements overlaid on the live image are linked to their corresponding semantic elements. To this end, nodes and text share a common property: *semanticExpression*; which indicates the expression to be evaluated during the representation of those graphical elements to obtain the correct semantic elements. Edges bind two nodes, so they have *semanticExpression* for source and destination elements. To define semantic expressions a query language, such as SQL for relational databases, AQL for EMF models, OCL for UML models, XPath/XQuery for XML files, JPQL for relational databases in Java, or LINQ for data querying in .NET, is required. Also, elements representing nodes, edges and labels share a common feature: *semanticCondition*; to set visual properties depending on the status of the model elements.

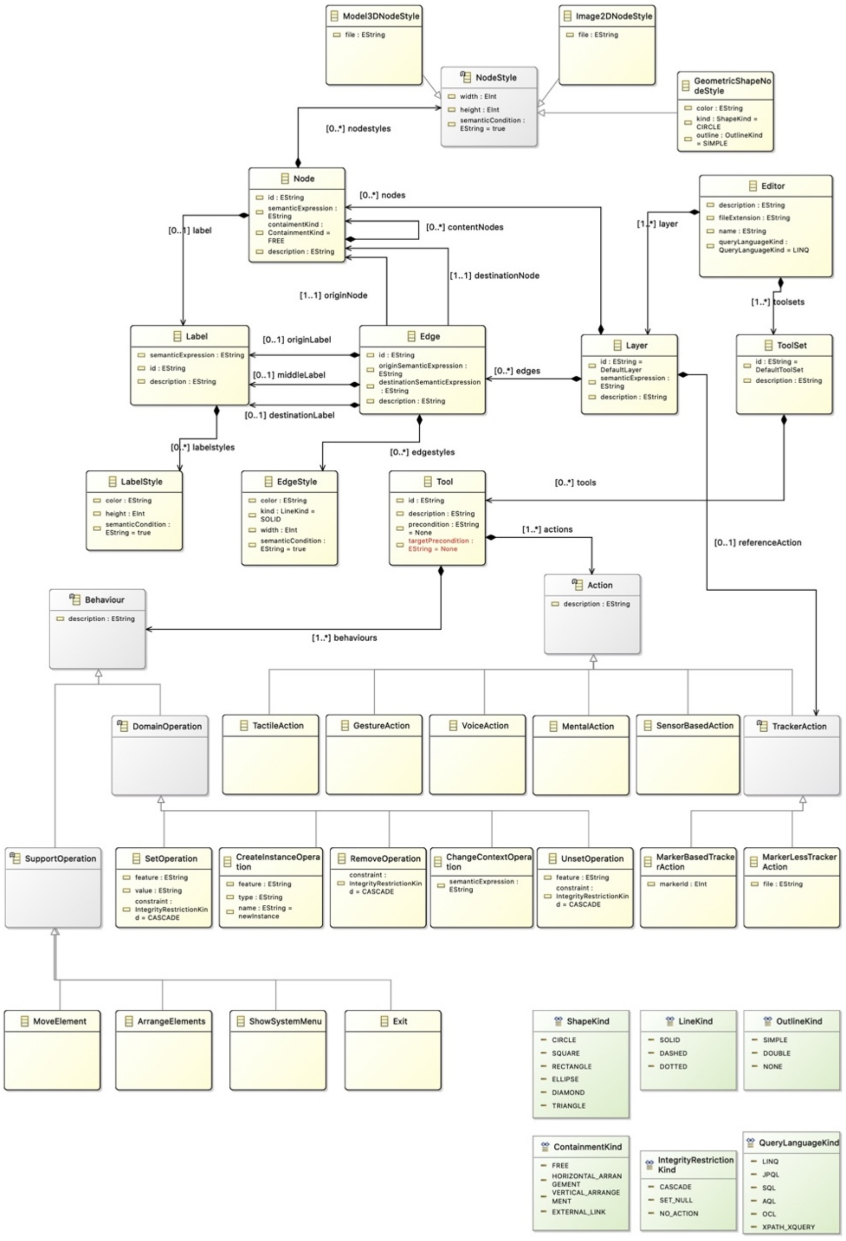


Fig. 1. Augmented Reality Editor metamodel.

A toolset is composed of *Tools* to manipulate the model elements overlaid on the real-world environment. A tool defines a correspondence between one or more *Actions* and one or more operation *Behaviours*. In AR applications, several tracking systems can be used: *Sensor-based* tracking, which uses the mobile device sensors, such as GPS, accelerometers and gyroscopes, to determine the location, orientation and direction of the device; *Marker-based* tracking, which requires labels containing a particular pattern; or *Markerless,* via image recognition systems to identify a location or constructing a map of an unknown environment [23], among others. In this regard, the tools to manipulate the models work with any of the trackers described. Furthermore, the editor layer has a reference action to correctly position the model in the real world and superpose all the virtual elements on the live image.

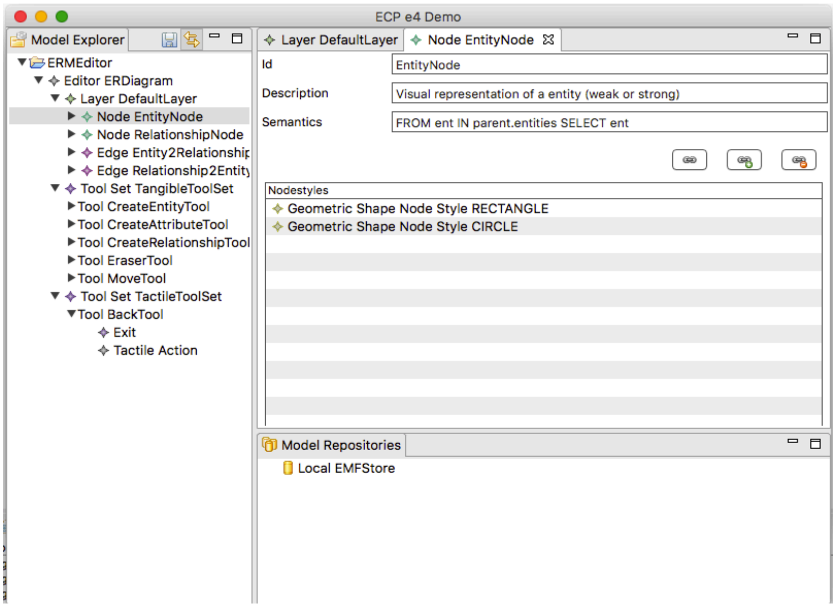
The metamodel has been designed to also manage multiple types of interactions, such as *Tactile Actions*, *Gesture Actions*, *Voice Actions*, and *Mental Actions*. The tools are linked to a set of *Domain Operations* which works on top of a given semantic model. These operations allow the editors to *Create* or *Remove instances*, *Set* or *Unset* features of the instances, and so forth. Some conventions were included to reduce the number of operations to declare. For example, *RemoveOperation* has a constraint to define the behaviour when a created node is deleted or updated, namely, remove children on cascade, set null or doing nothing. In addition, the semantic elements on which tools are used can be restricted by defining a precondition for the source and target elements. Besides the domain operations, there are other types of *Support operations* for *Moving* or *Arranging* virtual elements as well as *Showing the system menu* and *Exiting*.

3.3 Tools

During the development of the tools two strategies were considered: 1. developing an app able to interpret any model definition of an AR-based editor; 2. developing a common code base and through a prior code generation step, producing the language-specific code. The first option is supposed to be more flexible and easier to maintain because re-compilations are not required to parse new languages. However, we opted for the second to obtain better performance and to allow DSL developers to include ad-hoc modifications to the look and feel of the generated apps.

An extensible AR model editor. ARE4DSL, a mobile app for Android devices, is currently available to edit models using AR features. This software is capable of supporting different DSLs as long as additional source artifacts have been incorporated beforehand. ARE4DSL is developed in C# with Unity [56] and Vuforia [47] (a common AR development kit). Like any other modeling tool ARE4DSL allows users to open, create and edit models, as well as, export and import models in XML files. However unlike traditional visual modeling tools, ARE4DSL does not provide a canvas for drawing because any real surface can become a modeling area. Thus users only have to use the device screen for typing purposes, such as writing the model name or defining the value of a given property because the application can render the visual elements associated with all the semantic elements of the models. There is no toolbox with buttons on the screen. Instead, the operations mapped to the tools execute when the camera of the device recognizes the trackers defined by the DSL designer. In addition to these tools, a markerless auxiliary tool is provided to act as a virtual pointer on the scene. With this tool, a properties window will appear on the screen when the virtual pointer is hovering a certain element of the model.

Generating custom editors. The base application described above must be extended to give support to new DSLs. The abstract syntax will be defined with the standard EMF editor of Eclipse whereas the concrete syntax will be designed with a new Eclipse component specifically created for modeling AR editors (see Fig. 2). Once the abstract syntax and the concrete syntax of the language have been designed, a code generation step must be performed. To this end, an Acceleo module and a set of code templates were developed to generate the C# source code components required by the base tool to support a new modeling language. In this code, nodes and edges from the concrete syntax are mapped to *GameObjects*, the object containers of Unity to represent the visual elements. These objects enable the creation of the components to be rendered on the scene. Tools based on tracking actions are mapped to a set of targets included in a specific image database for Vuforia. Finally, during the app runtime, several C# scripts are in charge of detecting collisions among virtual elements to ensure the validation rules of the user model. The generated code has to be later placed into the *Resources/Scripts* directory of the base application. Finally, the whole source code (base app code + DSL-specific code) has to be compiled and built into the target platform, namely Android SDK by following the guidelines provided by Unity.



- ERMEditor
  - Editor ERDiagram
    - Layer DefaultLayer
      - Node EntityNode
      - Node RelationshipNode
      - Edge Entity2Relationship
      - Edge Relationship2Entity
    - Tool Set TangibleToolSet
      - Tool CreateEntityTool
      - Tool CreateAttributeTool
      - Tool CreateRelationshipTool
      - Tool EraserTool
      - Tool MoveTool
    - Tool Set TactileToolSet
      - Tool BackTool
        - Exit
        - Tactile Action

Id	EntityNode
Description	Visual representation of a entity (weak or strong)
Semantics	FROM ent IN parent.entities SELECT ent

Nodestyles
- Geometric Shape Node Style RECTANGLE
- Geometric Shape Node Style CIRCLE

Fig. 2. Tool for designing AR model editors.

From the user's perspective, two Eclipse IDE plug-ins have been developed. One plug-in provides the features to model the AR-based editors according to the metamodel previously described. This plug-in generates models with *.are* file extension. The second plug-in extends the options menu in the contextual menu associated to the *.are* files to launch the code generation. At the time of writing, ARE4DSL presents some limitations, namely it is only able to support tools based on AR markers and only supports LINQ as a language for defining semantic expressions.

4 Evaluation

In the previous section a framework for creating AR editors for DSLs was described. This framework provides a method, a reference model and a set of support tools and is available at the ARE4DSL website [53]. Below, an Entity-Relationship (ER) model editor featured with AR is illustrated, as well as a usability test carried out with users.

4.1 An Entity-Relationship Model Editor Featured with AR

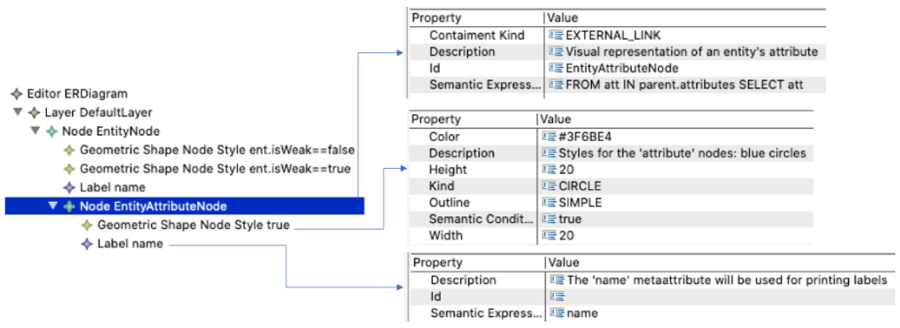
The technical feasibility of the framework is demonstrated by implementing the method and the tools provided for one common model in Software Engineering, widely used to model databases. Thus, this use scenario aims to show the design process required to build an editor of ER models.

Language design. The abstract syntax of this language has been designed with the EMF editor provided in Eclipse and is depicted using the Xcore [18] notation in Listing 1. According to this design, an ER model has a name and contains a series of entities and relationships. Each entity also has a name and a set of attributes. Entities that cannot be uniquely identified by its attributes are set with the *weak* flag. Relationships have a name and two references to the source and target entities, respectively. The cardinalities for both edges are defined through an enumerated value. Finally, attributes are defined by a name, a data type (from an enumeration), and two flags for tagging the attributes as primary keys and/or foreign keys. Note that this language does not define any link between attribute with the foreign key property and the corresponding relationship.

The concrete syntax of our language has been defined according to the metamodel of editors based on AR. Figure 3 shows a snippet of the designed model focusing on the visualization of attributes belonging to a given entity as blue (#3F6BE4) circles. Additionally, Figure 4 shows the set of actions of a tool based on an AR marker (ID 2) to trigger the creation of a new attribute on both entities and relationships. In these figures, it is noted how through the description metaproperty, DSL developers can document the elements of the concrete syntaxes.

Lst. 1. A simplified metamodel for Entity-Relationship models.

```
class ERModel {
       contains Entity[] entities
       contains Relationship[] relationships
       id String name
}
class Entity {
       id String name
       contains Attribute[] attributes
       boolean isWeak
}
class Relationship {
       String name
       CARDINALITY sourceCardinality
       CARDINALITY targetCardinality
       refers Entity[1] source
       refers Entity[1] target
       contains Attribute[] attributes
}
class Attribute {
       String name
       DATATYPE dataType
       boolean isForeignKey
       boolean isPrimaryKey
}
enum DATATYPE {
       INTEGER = 0
       DATE = 1
       FLOAT = 2
       STRING = 3
}
enum CARDINALITY {
       ZERO_ONE = 0
       ZERO_MANY = 1
       ONE = 2
       ONE_MANY = 3
}
```



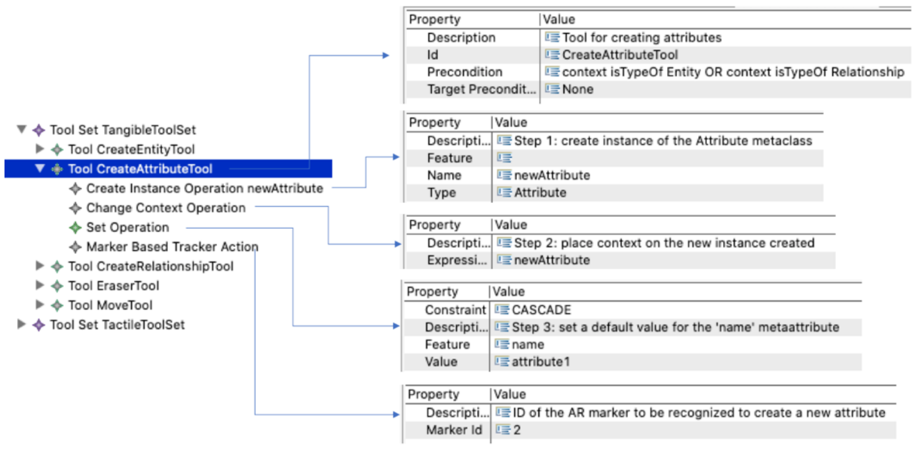
```
Editor ERDiagram
  Layer DefaultLayer
    Node EntityNode
      Geometric Shape Node Style ent.isWeak==false
      Geometric Shape Node Style ent.isWeak==true
      Label name
    Node EntityAttributeNode
      Geometric Shape Node Style true
      Label name
```

Property	Value
Contaiment Kind	EXTERNAL_LINK
Description	Visual representation of an entity's attribute
Id	EntityAttributeNode
Semantic Express...	FROM att IN parent.attributes SELECT att

Property	Value
Color	#3F6BE4
Description	Styles for the 'attribute' nodes: blue circles
Height	20
Kind	CIRCLE
Outline	SIMPLE
Semantic Condit...	true
Width	20

Property	Value
Description	The 'name' metaattribute will be used for printing labels
Id	
Semantic Express...	name

Fig. 3.. Node definition for displaying attributes as colored circles.



- Tool Set TangibleToolSet
  - Tool CreateEntityTool
  - Tool CreateAttributeTool
    - Create Instance Operation newAttribute
    - Change Context Operation
    - Set Operation
    - Marker Based Tracker Action
  - Tool CreateRelationshipTool
  - Tool EraserTool
  - Tool MoveTool
- Tool Set TactileToolSet

Property	Value
Description	Tool for creating attributes
Id	CreateAttributeTool
Precondition	context isTypeOf Entity OR context isTypeOf Relationship
Target Precondit...	None

Property	Value
Descripti...	Step 1: create instance of the Attribute metaclass
Feature	
Name	newAttribute
Type	Attribute

Property	Value
Descripti...	Step 2: place context on the new instance created
Expressi...	newAttribute

Property	Value
Constraint	CASCADE
Descripti...	Step 3: set a default value for the 'name' metaattribute
Feature	name
Value	attribute1

Property	Value
Descripti...	ID of the AR marker to be recognized to create a new attribute
Marker Id	2

Fig. 4.. Tool definition for triggering the creation of attributes through a given AR marker.

Editor creation and semantics definition. The abstract and concrete syntax models are injected to Acceleo plug-ins to automatically generate the components to be later included in the Unity base code. As a result, an Android mobile app is produced to manage ER models (see Figure 5). A video of the running tool is available online [54]. These models can be manipulated by using a set of AR markers previously customized with meaningful printed images. These markers were set as sides of a handier cube. The cube includes markers for creating entities, relationships between entities, and entity or relationship attributes, as well as markers for removing an element, moving elements and opening the preferences window. The app was also enriched with a speech recognizer module to allow users to define the names of the model elements with their own voice. Finally, in order to illustrate further possibilities of the model-driven approach, the language semantics was added by including a model-to-text transformation in this app. This transformation automatically produces SQL database definition scripts from the ER models designed using augmented reality.

4.2 Usability Test

To assess the applicability of the generated ER model editor, a usability evaluation [50] with users was conducted to measure the effectiveness, efficiency and satisfaction with the tool.

Study design. The experience was conducted in the context of the *20th International Symposium on Computers in Education* (SIIE) with a group of experts (N=11) on computer education. The participants in this study were first informed of the overall objectives of the tool and provided with general user guidelines. Then, they were tasked with developing an entity-relationship model with the tool. The participants were asked to create a model with a relationship between two entities, both of those containing two attributes. Once they completed the task, they took a survey. The research question was therefore: how easy is to create entity-relationship models with the tool provided?

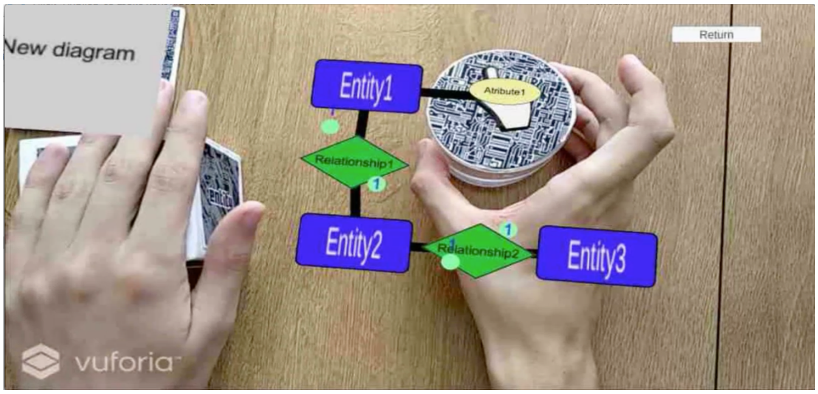


Fig. 5. An snapshot of the AR editor for ER models.

A Google Forms questionnaire, accompanied by a consent and revocation form to guarantee the privacy of personal data, was designed for the survey. The questionnaire included profiling questions such as gender, age, education, years of experience doing modelling tasks in Software Engineering and the date when they last created an entity-relationship model. There were also 10 questions (see Table 1) related to the user experience with the tool to be answered with a five point likert scale (from strongly agree to strongly disagree) according to the System Usability Scale (SUS) [10]. In addition, free text questions were included to gather qualitative feedback.

Table 1. System Usability Scale questionnaire.

Question	Avg (std.dev.)
I think that I would like to use the tool in my next modeling task	1.91 (1.10)
I found the tool unnecessarily complex	1.45 (1.20)
I think the tool is easy to use	2.91 (0.89)
I think that I would need the support of a technical person to be able to use this system	1.64 (1.12)
I found the various functions in the tool were well integrated	2.91 (0.70)
I thought there was too much inconsistency in the app	0.82 (0.46)
I would imagine that most researchers would learn to use this app very quickly	2.91 (0.70)
I found the tool very cumbersome to use	2.09 (1.37)
I felt very confident using the app	3.00 (0.77)
I needed to learn a lot of things before I could get going with this system	0.82 (0.87)
	2.04 (0.86)

Data compilation and analysis. Each response was turned into a numerical score between 0 (strongly disagree) and 4 (strongly agree). Although the sample is not large (N=11), it reveals interesting results. On average (see Table 1), participants neither agreed nor disagreed (avg. 2.04, std.dev. 0.86) regarding the questions proposed. However, we can note some insights, such as the participants felt confident using the app, they did not appreciate much inconsistency in the app, and they did not need to learn a lot of things before they could get going with the system. Then, the scores were standardized and aggregated using the formula below, with odd questions are positive (P), whereas even ones are negative (N).

$$\sum_{i=1}^{5}((P_i * 2.5) - 2.5) + \sum_{i=1}^{5}(12.5 - N_i * 2.5) . \quad \textbf{(1)}$$

Concerning the research question, the average score of the usability of the tool was 67.05 with a standard deviation of 13.73 in a range from 0 (negative) to 100 (positive). This value reflects a fair/high usability grade but is close to a good usability level (70) according to the interpretation of Bangor et al. [8]. Also, in this study, gender influences the perceived usability, since the results were better for females (N 5, avg. 74, **std. dev**. 13.76) than males (N 6, **avg**. 61.25, **std. dev**. 11.69).

In addition to the quantitative assessments, participants were asked to provide feedback on the main strengths and weaknesses observed in the tool and suggestions for improvement. According to the users, the main strengths of the tool with regard to traditional modeling tools were its novelty and its potential to interact differently. Some of them considered it as an innovative, motivating and enjoyable software, and others perceived it as a good tool for teaching databases to novices. However, some participants considered some of the interaction gestures somewhat complicated, and others noticed the need for having a big, clear area to easily work with the system. Furthermore, one of the participants did not find AR adequate for such an abstract field as software modeling. The experts provided us with suggestions for improvement, such as avoiding pressing the microphone button for activating the voice input by using a wake word, improving response time of the app for a smoother experience and establishing a direct connection to a database management system for the automatic generation of the scheme.

5 Discussion

The main contribution of this research is to provide a framework for creating AR model editors with reduced effort. We achieve that by following a model-driven development approach. However, several issues related to usability, scalability, maintainability and portability of the generated editors must be considered.

Regarding the usability of the generated AR editors, in their current form, they may be not so friendly as mature commercial/open source modeling tools. However, we continue working in polishing their interfaces to create a richer experience from user feedback we gathered. AR editors provide with a style of interaction used in a standard 2D editor, but in a 3D space. This is because the current implementation only works using AR trackers, with one of them representing a virtual pointer on the scene. However, the framework for the automated generation of AR editors allows DSL designers to define more diverse interaction mechanisms, such as voice-based actions and hand gestures. Thus, users can leverage some of their bodily-kinesthetic abilities.

We have presented an evaluation scenario consisting on the development of an AR-enriched ER editor consisting of a few models in a learning scenario. However, in realistic scenarios users can deal with larger models. Regarding the possible scalability issue with more complex scenarios, even for common 2D DSL editors it is laborious to deal with large models. To cope with that issue, we are working on a solution based on filters, which are well-known capabilities in several visual tools.. Filters can be enabled or disabled dynamically by the user to show or hide certain elements so that he/she will be able to do it through alternative interaction modes, which is an advantage over what regular visual tools provide to manage complex models.

Another aspect of the generated editors is related to their maintainability. Since we opted for a code-generation step from a model instead of using a model-interpretation strategy, the code obtained can be later programmatically adapted to particular users' needs.

Finally, novel AR engines such as ARKit or ARCore are emerging to create richer experiences. The current implementation of ARE4DSL uses the Vuforia AR engine, however, it can be migrated without much effort, since the method and the PIM model described in this paper are agnostic of the low-level technology used.

6 Conclusion

Bodily-kinesthetic abilities are not currently supported in common modeling tools. Nowadays, model editors for both general and domain languages rely only on the visual-spatial and logical-mathematical abilities of users. This research promotes the development of tangible modeling user interfaces, which take advantages of multimodal interactions and AR. Since the above technologies have already been successfully used for educational purposes, tangible DSL editors can also help users with different learning styles to train their modeling skills.

This paper presents a framework consisting on a systematic method, a metamodel for defining multi-modal editors and some supporting tools. The method adapts the general phases of DSL development to follow a model-driven process namely: abstract syntax definition with Ecore; concrete syntax definition with the new metamodel proposed; editor generation with the ARE4DSL tool; and optionally, definition of the language semantics. As a result, DSL designers will be able to design the model editors to be run on mobile devices without any concern about low-level details of the technological stack. The framework has been only used for developing editors for two kinds of models in Software Engineering: state machine models and entity-relationship models; which may be a concern regarding its validity.

A usability test was conducted on the entity-relationship model editor that was developed with the framework proposed in this work. The study shows that, despite some limitations, the tool has potential to support the teaching of modeling and work with entity-relationship models in an innovative way. The perceived usability was fair/high and is better appreciated by females and people with more experience in modeling tasks. The SUS, a reliable and common tool to measure usability, was used to ensure the construct validity of our test. In addition, the internal validity is guaranteed by the use of simple techniques and visual representations for the analysis of the data. There are no apparent threats to the external validity of the study because the presented findings are not conclusive nor generalisable due to the small sample, requiring hence the replication of the study with more users as well as performing a deeper heuristic evaluation.

In addition to the expected usability improvements identified after the evaluation, the ARE4DSL tool will be extended to support other interaction methods –already considered in the framework, such as the verbal and gestural ones, by leveraging the background acquired during the development of VEDILS [44], a visual tool for generating multimodal applications. Additionally, a migration tool to transform existing visual DSLs created with GMF or Sirius into AR-based alternatives will be developed. Having a development environment which helps users to edit models is essential to improve modeling abilities and boost productivity. Thus, tangible DSLs enhanced with AR or VR features can support software modeling tasks.

Availability of data and materials. The support tools provided with this paper and the datasets generated and analysed during the current study are available at the ARE4DSL website [https://github.com/spi-fm/ARE4DSL]. The Google Forms questionnaire for the survey is available at [https://tinyurl.com/y8mclu5p].

Acknowledgments. This work has been developed in the VISAIGLE project, funded by the Spanish National Research Agency (AEI) with ERDF funds under grant ref. TIN2017-85797-R and supported by the University of Cadiz with grant ref. EST2018-167

References

1. Abrahão S, Bordeleau F, Cheng B, Kokaly S, Paige RF, Störrle H, Whittle J. User experience for model-driven engineering: Challenges and future directions. In: 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems, IEEE, pp 229–236 (2017)
2. Agner LTW, Lethbridge TC, Soares IW. Student experience with software modeling tools. Software & Systems Modeling 18(5):3025–3047 (2019)
3. Aguilera González NA. Development of spatial skills with virtual reality and augmented reality. International Journal on Interactive Design and Manufacturing 12(1):133–144 (2018)
4. Azuma RT. A survey of augmented reality. Presence: Teleoperators and Virtual Environment 6(4):355–385 (1997)
5. Bacca J, Baldiris S, Fabregat R, Graf S, et al. Augmented reality trends in education: a systematic review of research and applications. Journal of Educational Technology & Society 17(4):133 (2014)
6. Bain K, Basson SH, Wald M. Speech recognition in university classrooms: liberated learning project. In: Proceedings of the fifth international ACM conference on Assistive technologies, ACM, pp 192–196 (2002)

7. Bang G, Yang J, Oh K, Ko I. Interactive experience room using infrared sensors and user's poses. Journal of Information Processing Systems 13(4):876–876 (2017)
8. Bangor A, Kortum P, Miller J. Determining what individual SUS scores mean: Adding an adjective rating scale. Journal of Usability Studies 4(3):114–123 (2009)
9. Brambilla M, Cabot J, Wimmer M. Model-driven software engineering in practice. Morgan & Claypool Publishers, California, USA (2012)
10. Brooke J. SUS: a retrospective. Journal of Usability Studies 8(2):29–40 (2013)
11. Budinsky F. Eclipse modeling framework: a developer's guide. Addison-Wesley Professional, Boston, USA (2004)
12. Catala, A., Jaen, J., Martinez-Villaronga, A. A., & Mocholi, J. A. AGORAS: Exploring creative learning on tangible user interfaces. In 2011 IEEE 35th Annual Computer Software and Applications Conference (pp. 326-335). IEEE. (2011)
13. Corral JMR, Balcells AC, Estévez AM, Moreno GJ, Ramos MJF. A game-based approach to the teaching of object-oriented programming languages. Computers & Education 73:83–92 (2014)
14. Daniela L, Lytras MD. Themed issue on enhanced educational experience in virtual and augmented reality. Virtual Reality pp 1–3 (2019)
15. Di Serio Á, Ibáñez MB, Delgado Kloos C. Impact of an augmented reality system on students' motivation for a visual art course. Computers & Education 68:586 – 596 (2013)
16. Eclipse Foundation. Acceleo. http://www.eclipse.org/acceleo
17. Eclipse Foundation. Graphical Modeling Framework. http://www.eclipse.org/modeling/gmp/
18. Eclipse Foundation Xcore. https://wiki.eclipse.org/Xcore
19. Esteban G, Fernández C, Conde MA, García-Peñalvo FJ. Playing with SHULE: surgical haptic learning environment. In: Proceedings of the Second International Conference on Technological Ecosystems for Enhancing Multiculturality, ACM, pp 247–253 (2014)
20. Eysholdt M, Behrens H. Xtext: implement your language faster than the quick and dirty way. In: Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, ACM, pp 307–309 (2010)
21. Follmer S, Leithinger D, Olwal A, Hogge A, Ishii H. inFORM: Dynamic physical affordances and constraints through shape and object actuation. In: 26th ACM Symposium on User Interface Software and Technology, vol 13, pp 417–426 (2013)
22. Fowler M. Domain-specific languages. Pearson Education, New Jersey, USA (2010)
23. Fuentes-Pacheco J, Ruiz-Ascencio J, Rendón-Mancha JM. Visual simultaneous localization and mapping: a survey. Artificial Intelligence Review 43(1):55–81 (2015)
24. Gardner HE. Intelligence reframed: Multiple intelligences for the 21st century. Hachette UK, New York, USA (2000)
25. Gerhart M, Boger M. Concepts for the model-driven generation of graphical editors in eclipse by using the graphiti framework. International Journal of Computer Techniques 3(4) (2016)
26. Green TR, Petre M. When visual programs are harder to read than textual programs. In: Human-Computer Interaction: Tasks and Organisation, Proceedings of ECCE-6 (6th European Conference on Cognitive Ergonomics). GC van der Veer, MJ Tauber, S. Bagnarola and M. Antavolits. Rome, CUD, pp 167–180 (1992)
27. Heldal R, Pelliccione P, Eliasson U, Lantz J, Derehag J, Whittle J. Descriptive vs prescriptive models in industry. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, ACM, pp 216–226 (2016)
28. Ishii H, Ullmer B. Tangible bits: towards seamless interfaces between people, bits and atoms. In: Proceedings of the ACM SIGCHI Conference on Human factors in computing systems, ACM, pp 234–241 (1997)
29. Jessen, A., Hilken, T., Chylinski, M., Mahr, D., Heller, J., Keeling, D. I., & de Ruyter, K. The playground effect: How augmented reality drives creative customer engagement. Journal of Business Research, 116, 85-98. (2020)

30. Jetbrains. Meta programming system. https://www.jetbrains.com/mps/
31. Jouault F, Allilaire F, Bézivin J, Kurtev I, Valduriez P. ATL: a QVT-like transformation language. In: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, ACM, pp 719–720 (2006)
32. Kang J. Effect of interaction based on augmented context in immersive virtual reality environment. Wireless Personal Communications 98(2):1931–1940 (2018)
33. Kelly S, Tolvanen JP. Collaborative creation and versioning of modeling languages with metaedit+. In: Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, ACM, New York, USA, pp 37–41 (2018)
34. Kleppe AG, Warmer JB, Bast W. MDA explained: the model driven architecture: practice and promise. Addison-Wesley Professional, Boston, USA (2003)
35. Klint P, van der Storm T, Vinju J. EASY Meta-programming with Rascal, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 222–289 (2011)
36. Kolovos DS, García-Domínguez A, Rose LM, Paige RF. Eugenia: towards disciplined and automated development of GMF-based graphical model editors. Software & Systems Modeling 16(1):229–255 (2017)
37. Kosar T, Bohra S, Mernik M. Domain-specific languages: A systematic mapping study. Information and Software Technology 71:77–91, (2016)
38. Liou HH, Yang SJ, Chen SY, Tarng W. The influences of the 2D image-based augmented reality and virtual reality on student learning. Journal of Educational Technology & Society 20(3):110–121 (2017)
39. López-Fernández JJ, Garmendia A, Guerra E, de Lara J. An example is worth a thousand words: Creating graphical modelling environments by example. Software & Systems Modeling 18(2):961–993 (2019)
40. McNerney TS. From turtles to tangible programming bricks: explorations in physical language design. Personal and Ubiquitous Computing 8(5):326–337 (2004)
41. Medina Herrera L, Castro Pérez J, Juárez Ordóñez S. Developing spatial mathematical skills through 3D tools: augmented reality, virtual environments and 3D printing. International Journal on Interactive Design and Manufacturing (2019)
42. Microsoft. Visual Studio. https://www.visualstudio.com/
43. Milgram P, Kishino F. A taxonomy of mixed reality visual displays. IEICE Transactions on Information and Systems 77(12):1321–1329 (1994)
44. Mota JM, Ruiz-Rube I, Dodero JM, Person T, Arnedillo-Sánchez I. Learning analytics in mobile applications based on multimodal interaction. In: Software Data Engineering for Network eLearning Environments, Springer, Cham, Switzerland, pp 67–92 (2018)
45. Müller S, Würsch M, Schöni P, Ghezzi G, Giger E, Gall HC. Tangible software modeling with multi-touch technology. In: Proceedings of the 5th International Workshop on Co-operative and Human Aspects of Software Engineering, pp 100–104 (2012)
46. Navarro-Prieto R, Cañas JJ. Are visual programming languages better? the role of imagery in program comprehension. International Journal of Human-Computer Studies 54(6):799 – 829 (2001)
47. PTC Vuforia. https://www.ptc.com/en/products/augmented-reality
48. Regenbrecht HT, Wagner M, Baratoff G. MagicMeeting: A collaborative tangible augmented reality system. Virtual Reality 6(3):151–166 (2002)
49. Rodgers DL, Withrow-Thorton BJ. The effect of instructional media on learner motivation. International Journal of Instructional Media 32(4):333 (2005)
50. Rubin J, Chisnell D. Handbook of usability testing: how to plan, design and conduct effective tests. John Wiley & Sons, New York, USA (2008)
51. Rumpe B, Hölldobler K. Monticore 5 language workbench. edition 2017. Shaker Verlag, Aachen, Germany (2017)

52. Sargent R, Resnick M, Martin F, Silverman B. Building and learning with programmable bricks. In: Kafai YB, Resnick M (eds) Constructionism in Practice. Designing, Thinking, and Learning in A Digital World, Routledge, New York, USA, pp 161–173 (1996)
53. SPI-FM Group ARE4DSL framework. https://github.com/spi-fm/ARE4DSL
54. SPI-FM Group ARE4DSL video. https://www.youtube.com/watch?v=DMU6TSxY5DE
55. Stopczynski A, Stahlhut C, Petersen MK, Larsen JE, Jensen CF, Ivanova MG, Andersen TS, Hansen LK. Smartphones as pocketable labs: Visions for mobile brain imaging and neurofeedback. International Journal of Psychophysiology 91(1):54–66 (2014)
56. Unity Technologies Unity. https://unity3d.com
57. Viyovic V, Maksimovic M, Perisic B. Sirius: A rapid development of DSM graphical editor. In: IEEE 18th International Conference on Intelligent Engineering Systems INES 2014, IEEE, pp 233–238 (2014)
58. Wachsmuth GH, Konat GD, Visser E. Language design with the spoofax language workbench. IEEE software 31(5):35–43 (2014)
59. Wang D, Zhang Y, Chen S. E-Block: A tangible programming tool with graphical blocks. Mathematical Problems in Engineering 2013 article ID 598547 (2013)
60. Wei L, Zhou H, Soe AK, Nahavandi S. Integrating kinect and haptics for interactive STEM education in local and distributed environments. In: IEEE/ASME International Conference on Advanced Intelligent Mechatronics, IEEE, pp 1058–1065 (2013)
61. Whittle J, Hutchinson J, Rouncefield M. The state of practice in model-driven engineering. IEEE software 31(3):79–85 (2014)