1. INTRODUCTION

A modeling language (ML) is a set of words and symbols, supported by validation rules and semantics, which make it possible to create models or diagrams. It can be graphical/visual or textual, depending on its scope and domain. An ML might be classified as general-purpose (GPML) or domain-specific modeling language (DSML) [van Deursen et al. 2000, Luoma et al. 2004, Mernik et al. 2005, Kosar et al. 2010]. A GPML is characterized by having a greater number of generic constructs, which encourages a wider and widespread use in different fields of application. UML or SysML are popular examples of GPMLs by providing large sets of constructs and notations used for specifying and documenting, respectively, software systems according to the object-oriented paradigm, or for system engineering. On the other hand, DS(M)Ls tend to use few constructs or concepts that are closer to its application domain.

Since a DS(M)L is expressed using domain concepts, it is normally easier to read, understand, validate and communicate with, facilitating cooperation between developers and domain experts. Moreover, some argue that DS(M)Ls can improve productivity, reliability, maintainability and portability [van Deursen et al. 2000]. However, the use of a DS(M)L can raise some problems, such as the cost of learning, implementing and maintaining a new language, as well as the support tools to develop with it [Mernik et al. 2005].

Either for GPMLs as for DSMLs, nowadays there is a great variety of modeling languages. For instance, BPMN is specific to business process design although UML’s activity diagram is also adequate for this purpose [Object Management Group 2013, Object Management Group 2011], DEMO is specific to enterprise architecture [Dietz 2001], XIS to Interactive Applications, and Petri Nets to Distributed Systems [Desel and Juhás 2001]. Since there are more languages and approaches than domains, this results in overlapping effort for researchers and disorientation for modelers.

In order to help professionals selecting the most suitable modeling language to work on so many specific and different contexts, evaluation frameworks are needed. An evaluation framework is a set of properties, metrics, concepts and other parameters that compare and assert the languages’ characteristics. For this work, we have proposed ARENA Framework to enlighten individuals that work with models on this area. Our goal in this article is to help the user choosing the most appropriate modeling language, in order to assure a quality output, to feel more supported and to work in a more rational way.

In this work we propose a framework to evaluate and compare the quality of User-Interface Modeling Languages (UIMLs) and Business Process Modeling Languages (BPMLs), focused on either their general and specific characteristics. The definition of a modeling language involves multiple
aspects or facets which have to be taken into consideration, namely abstract syntax, concrete syntax, and semantics.

On one hand, UIMLs are DSMLs that are specifically used for modeling the user interface of desktop, web or mobile applications, supporting the design and implementation phases [Achilleos et al. 2008]. They denote conceptual abstractions that are present on user interfaces [Frank 1996].

On the other hand, BPMLs are DSMLs that contain concepts and graphical elements which allow the user to design business process in virtually all industry areas e.g., banking, insurance and transports, but it is also used for academic purposes. They shall allow traceability and hierarchy between processes and diagrams [Nysetvold and Krogstie 2005].

This extended abstract is organized in 6 sections: Section 1 introduces the context of this work and a summarized description of the problem and the proposed goals. Section 2 regards the background of this research, that includes definitions of the most important concepts for this work (modeling language, domain, user-interface modeling language, business process modeling language, model driven software development and quality). Section 3 presents the most important steps taken in the topic of quality on modeling, namely developed frameworks, community initiatives and projects that contributed to solve modeling problems. Section 4 proposes ARENA's Framework and Website as the solution to the previously referred problem. Section 5 compares, evaluates and discusses the quality of four UIMLs and three BPMLs with ARENA’s properties and metrics. Finally, section 6 shows the conclusions taken and some directions that this work may follow.

2. BACKGROUND

This section introduces the background of this work, namely the definition of its core concepts: Modeling Language, Domain, User-Interface Modeling Language, Business Process Modeling Language, Model-Driven Software Development and Quality.

2.1 Modeling Languages

According to section 1, a Modeling Language, which is a set of words and symbols, is created from a metamodel within a certain domain, therefore, it can be seen as a model of a modeling language [Ma et al. 2004]. In order to give the modeling languages functionality to use and evaluate the models, different types of mechanisms are included in the metamodeling process (algorithms, generic mechanisms, specific mechanisms and hybrid mechanisms) [Karagiannis and Kühn 2002]. In other words, the metamodel is able to highlight the properties of the model, derived from its capacity of abstraction. There are several metamodeling approaches. The most commonly used is Meta Object Facility (MOF) and it has been used, for instance, in the development of UML and SysML.

In terms of representation type, MLs can be divided in two groups: graphical and textual. On one hand, graphical MLs use a diagram technique with named symbols that represent concepts and lines that connect the symbols and represent relationships and various other notation to represent constraints. On the other hand, textual MLs use standardized keywords accompanied by parameters or natural language terms and phrases to make computer-processable expressions. Considering modeling approaches, in the field of computer science, more specific types of MLs have recently emerged, due to new challenges on very different areas, namely: Algebraic, Behavioural, Discipline-Specific, Domain-Specific, Framework-Specific, Object Oriented or Virtual Reality.

An ML is very important because it helps to elucidate their stakeholders that have modeling expertise to understand what and how they can make a representation of the system-of-interested. Also, its features such as syntax, abstraction and compatibility with programs can facilitate its use
and its correspondent metamodels may give a perception about the adequacy of the language to the concrete situation.

2.2 Domain

A domain (also referred as Problem Domain) is an engineering term referring to all information that defines the problem and constrains the solution (the constraints being part of the problem). It includes the goals that the problem owner wishes to achieve, the context within which the problem exists and all the rules that define essential functions or other aspects of any solution product. It represents the environment in which a solution must operate, as well as the problem itself\(^1\). In other words, it is any subset of a conception (being a set of elements) of the concrete or abstract universes, that is created as being some part or aspect of those universes [Hoppenbrouwers et al. 2005].

Also, it is the area of expertise or application that needs to be examined to solve a problem. A problem domain simply looks at only the relevant topics and excludes everything else.

This concept is important because the domain or the nature/context of the problem may influence the Modeling Language’s appropriateness for it.

2.3 User-Interface Modeling Language

As we already explained on Section 1, UIMLs are modeling languages that are used for designing the user interface of various platforms’ applications, supporting the design and implementation phases [Achilleos et al. 2008]. This work analyses and compares the main properties of four UIMLs, namely: UMLi [Silva 2002], UsiXML [Limbourg et al. 2004], XIS [Silva et al. 2003, Silva et al. 2007, Martins and Silva 2007] and XIS-Mobile [Ribeiro 2014, Ribeiro and Silva 2014a, Ribeiro and Silva 2014b]. We have selected them due to their support, complete documentation, project visibility and availability of their papers.

We are aware of other UIMLs such as DiaMODL [Tøttemeberg 2008], IFML [Object Management Group 2014], MARIA XML [Paternó 2009] or WebML [Ceri et al. 2002]. However, we acknowledge that they don’t fit the domain and our criteria as well as the previous four. WebML had strong influence in IFML, when Object Management Group created this language and that IFML’s most recent version (from 2014) is still in Beta. Therefore, we consider that there were too much similarities between both and comparing them would not prove effective. Regarding DiaMODL and MARIA XML, the first one is too focused on Dialog Modeling, Dataflow and State logic while the second is not quite appropriate due to its models’ development being highly dependable in Service-Oriented Architectures and Web Services, which implies exploiting annotations at design time and the language itself at runtime to support dynamic generation of user interfaces, thus limiting usability and analysis.

2.4 Business Process Modeling Languages

According to what we said in the previous section, BPMLs are modeling languages that allow business process designing in many different activity areas. Normally, they are Extensible Markup Language (XML)-based metalanguages, as a means of modeling business processes, much as XML is, itself, a metalanguage with the ability to model enterprise data. Some of their features are being able to list easily distinguishable concepts, include control, structure and cancellation patterns and hierarchical models [Nysetvold and Krogstie 2005].

In order to make our solution more useful and complete, we have chosen three modeling languages within this specific domain: UML, BPMN and DEMO.

---

\(^1\) “Problem Domain”, by Cunningham & Cunningham, Inc. (http://c2.com/cgi/wiki?ProblemDomain)
The Unified Modeling Language (UML) is a graphical language for specifying, constructing, and documenting the artefacts of systems. It is a general-purpose modeling language that can be used with all major object and component methods, and that can be applied to all application domains (e.g., health, finance, telecom, aerospace) and implementation platforms (e.g., J2EE, .NET) [Object Management Group 2011].

The Business Process Model and Notation (BPMN) is another graphical language, which main goal is to provide a notation that is easily understandable by all stakeholders, whom include business analysts that create the initial drafts of the processes, technical developers that are responsible for implementing the technology that will perform those processes, and finally, business people who will manage and monitor those processes [Object Management Group 2013].

DEMO (Design and Engineering Methodology for Organisations) is the leading methodology in the new discipline of Enterprise Engineering (EE). The theory of DEMO is that this social interaction takes place in universal patterns, called transactions. Business processes become clear tree structures of transactions, instead of mind-bending railroad yards [Dietz 2001, Dietz 2006].

2.5 Model-Driven Software Development

Model-Driven Software Development (often abbreviated to MDSD or MDD) consists in a software development process of designing models that represent how the system does what it is supposed to do. It is mainly focused on analyzing requirements and implementing model-to-model and model-to-code transformations in order to improve software productivity, process quality and the final output [Silva 2015]. It is used as well for software testing, business process development, software architectures and enterprise architectures.

As we can see in Figure 1, modeling is a learning process in which cooperative actors construct together their perception of reality [Hoppenbrouwers et al. 2005]. This approach is more efficient because it takes advantage of a professional, qualified and trustworthy team that uses communication and tool expertise to produce quality models that satisfy the requirements and may generate quality software.

2.6 Quality

Regarding a model that may generate software, its quality can be divided into internal and external [ISO/IEC 9126-1 2001].

Internal quality consists on the characteristics of the software product from an internal view. Internal quality requirements are used to specify properties of interim products. They can include static and dynamic models, other documents and source code [ISO/IEC 9126-1 2001].

External quality is defined by the characteristics of the software product from an external view. It is the quality of the executed software, which is typically measured and evaluated while testing in a
simulated environment with simulated data using external metrics. These metrics should be aligned with the external quality requirements, which goal is to specify the required level of quality from the external view [ISO/IEC 9126-1 2001].

Quality is not yet a common priority when developing a Domain-Specific Language (DSL). The focus is currently on getting up a systematic development of these Modeling Languages, a goal that has not yet been reached because it has been doing more study of the technical aspects of DSLs’ design and implementation, such as: case studies and technical reports on individual DSLs; design approaches and techniques for implementing DSLs; and integrating DSLs with other developmental approaches [Strembeck and Zdun 2009].

3. RELATED WORK

This section presents the most important projects developed in the context of quality on modeling and modeling languages, namely frameworks, community initiatives and other tools or papers that contributed to solve modeling problems. Consequently, it also intends to explain the type of design and functions that we wanted, when we developed the ARENA framework and, afterwards, launched the ARENA website.

3.1 CMA MODELS

The "Comparing Modeling Approaches" (also known as CMA MODELS) is an international annual event, started in 2011, that is comprised of presentations and discussion on modeling approaches and styles about the best solution to a case study which was presented at AOM Bellairs Workshop, which occurred on the same year [Capozucca et al. 2012].

The case study presents a Car Crash Management System and offers the choice of modeling either a single system or a software product line. The case study also includes a comparison criteria that is intended to help understanding, analyzing and comparing the presented approaches. They are divided into Modeling Dimensions and Key Modeling Concepts [Georg et al. 2012]. The first group is characterized to evaluate the approach by development phases, activities in which it is useful and languages and notations used (e.g. documents used for the assessment, which problems does the approach address, semantics etc.). The second group aims to classify the approach in terms of building blocks and attributes and identify qualities or improvements brought by the approach (e.g. Modularity, Traceability, Reduction of Modeling Effort etc.). The latter can be applied also to the models produced by the approach, although the focus is the approach itself [Capozucca et al. 2012].

3.2 SEQUAL

SEQUAL is a framework that was presented in 1994 with the goal of evaluating systematically the quality of information systems and other conceptual models through the notions of syntactic, semantic and pragmatic applied to them [Krogstie et al. 1995]. One year later it was extended to more three features, inspired by FRISCO’s six semiotic layers of communication. It is considered the first framework to have the objective on evaluating models’ quality.

Essentially, the first version of the framework only had four concepts — Model, Domain, Language and Audience interpretation — and six pair relations between them, being "semantic quality" (Model $\rightarrow$ Domain), "syntactic quality" (Model $\rightarrow$ Language), "pragmatic quality" (Model $\rightarrow$ Audience interpretation) and "appropriateness" on the three remaining relations (Domain $\rightarrow$ Language, Language $\rightarrow$ Audience interpretation and Domain $\rightarrow$ Audience interpretation). In Figure 2, we display its 1995 extension, which brought much more complexity: on one hand, the addition of a new entity (Participant knowledge) and the creation of four relationship types ("physical quality", "semantic quality", "syntactic quality" and "pragmatic quality"
"perceived semantic quality", "social quality" and "language quality"). On the other hand, Appropriateness disappeared due to its lack of specification.

In 2012, Krogstie makes a third instance of his framework and adds "deontic" as a new quality criteria that focuses on what is right and needed for the organization’s goals. The main changes are the shift of feasible validity and feasible completeness (former semantic quality metrics), feasible comprehension (former pragmatic quality metric) and feasible agreement (former social quality) to deontic quality along with the creation of feasible perceived validity and feasible perceived completeness [Krogstie 2012]. SEQUAL's formulas are based on the audience, the language, the model, the domain, the audience's knowledge and the audience's interpretation.

The SEQUAL framework contributed to this work because it is the first and main reference regarding frameworks that evaluate quality, in the MDSD context. It is an extensive developed work that also contemplates models’ quality and even evaluates UML and BPMN.

3.3 BPML Quality Assessment with Generic Framework

On 2005, a paper about focusing the standardization of business process modeling was presented. It intended to compare some Modeling Languages that could be used for this context. The three selected BPM languages were EEML, UML and BPMN, and their framework comprised the following items: Goals of modeling task, language extension, domain, externalized model, knowledge of the stakeholders, the social actors’ interpretation and the technical actors’ interpretation [Krogstie 2012].

3.4 Testbed project

On 1997, Teeuw and van der Berg published a paper about their perspective on general quality criteria for conceptual models and a framework that was used to evaluate the redesign of business processes. Like SEQUAL, they also defend that a good, quality model must have syntactic, semantic and pragmatic qualities and, considering that the concepts can be captured with a suitable language (assuring the first quality), they present as criteria for the second and the third: completeness, inherence, clarity, consistency, orthogonality and generality [Teeuw and van der Berg 1997]. Considering the Testbed evaluation framework, it was applied to behavior models and it focused on answering three questions: "which aspects of business process can be expressed?", "how easily can they be expressed?" and "when should we use specific models?".
Although they use interesting criteria, the presented frameworks don't satisfy the goal of this work because they evaluate other matters instead of UIMLs, respectively, the Quality of Conceptual Models, the Quality of Business Process and Business Process redesign, as we can see on Table 1.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Context</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQUAL</td>
<td>Definition of Systematic Quality</td>
<td>Focused on Conceptual Models, instead of Modeling Languages</td>
</tr>
<tr>
<td>BPML Quality</td>
<td>Criteria for Conceptual Models</td>
<td>Intends to analyse the Quality of Business Process, which is too specific</td>
</tr>
<tr>
<td>Assessment</td>
<td>Evaluation of Quality on three BPMLs</td>
<td>It is focused in Business Process Redesign, not Modeling Languages</td>
</tr>
<tr>
<td>Testbed project</td>
<td>Quality Assurance on Redesign of Business Process of Conceptual Models</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Summary of the related work frameworks

4. PROPOSED SOLUTION

This section presents our proposal to solve the problem of this thesis. Also, it describes its implementation on a web system, which intends to gather and accomplish the project’s purposes.

4.1 ARENA Framework

Since ARENA is a tool conceived to evaluate languages that allow concept representation, it needs a conceptual model of its own. Figure 3 shows that a Modeling Language's Quality is the central class and it contains the final quantitative output. This class uses the Modeling Language's information as an input and returns a Value within a range, as an output.
The latter is calculated using a formula that receives as an input the values of the quality and quantity evaluations (respectively represented as Dimension and Metric classes) and multiplies each by a previously defined Weight. The returned output is a Value that represents the language’s Quality within a rating scale, as explained hereinafter on Section 5. All domains share the General Properties, as opposite to Specific Properties, that differ from each other (e.g. UIMLs, BPMLs etc). The Notation Kind enumeration is being developed as a drop text, so the user can select one option from the displayed list. The proposed framework intends to bridge that gap and it is composed by general and specific properties, which are described in the following subsections.

4.1.1 General Properties

**Comprehension.** The language shall be able to precisely specify systems so that the developing team and the stakeholders (e.g. customers, operators, analysts, designers) can share their conceptions of reality and better comprehend the produced models. All team members have their unique perspective and interpretation of the real world and when it comes to achieve a common understanding, communication helps to mediate discussions, develop education and manage knowledge [Hoppenbrouwers et al. 2005]. Every participant must acknowledge his part of the externalized model, as we have seen on subsection 3.2. For instance, a designer shall be able to understand the concepts and rules of the language, in order to optimize his working time and effort to raise productivity [Teeuw and van der Berg 1997]. **Dimensions:** Communication and Expressiveness.

**Domain.** The environment where the language is going to be used to create models can facilitate or constrain its use [Nysetvold and Kroghstie 2005] and all the statements that are possible to make with it shall be correct and relevant to the problem’s resolution, within the domain. This will influence in the model’s validity [Krogstie et al. 1995]. This framework states representing the domain as domain appropriateness, i.e. the ability to describe the domain’s statements using the modeling language [Krogstie et al. 1995]. It should ideally be able to express things that are only in the domain but be powerful enough to include everything that is in the domain. In the best case scenario, there is no statement that can’t be expressed. The conceptual modeling activity must be adapted in the context in which models are going to be created [Hoppenbrouwers et al. 2005]. **Metric:** Ratio between Domain’s Number of Concepts and Language’s Meta-Model Number of Concepts.

**Functionality.** This property can be formally defined as the particular set of functions or capabilities associated with computer software or hardware or an electronic device, alternatively, the system’s behaviour quality [López et al. 2007]. We take into account if the ML has a list of simple and reusable templates that help the user solving common problems that appear during the design phase and if the compatible programs are capable of transforming the language’s models format into another format or if the language can produce more than one format. On one hand, and according to what we said in Section 2.5 about MDSD, it is crucial to evaluate the language’s ability or inability to generate textual artefacts from models and if so, which mechanisms or techniques make it possible. On the other hand, it is important to know which systems, applications or mechanisms can analyze the models and diagrams created by the user and validate them. **Metrics and Dimensions:** Pattern Usage, Tool Support - Model to Model (M2M) Transformations, Tool Support - Model to Text (M2T) Transformations and Tool Support - Validation.

**Interoperability.** The language must be fully compatible with several tools, i.e. should allow to do the same tasks and diagrams in different software tools. Also, it shall be possible to combine with another modeling and programming languages (PLs) and tools, being supported by mechanisms that guarantee those possibilities. **Metrics and Dimensions:** Number of Compatible Applications [ISO/IEC 25010.2 2008], Number of Integration Mechanisms [Karagiannis and Kühn 2002] and Compatibility.

**Maintenance.** The language must be appropriate to the present needs, that is, it should not have obsolete concepts or operations. Its concepts shall also be consistent and non-ambiguous, i.e., each one
of them must have only one meaning in the real world, when representing it [Teeuw and van der Berg 1997]. Also, if the generated models systematically don’t contain semantic quality (completeness and validity), it may imply that the language needs maintenance. Nevertheless, the language must have evolution and be flexible so that it is possible to add further elements or layers. **Dimensions**: Stability, Changeability, Consistency, Reusability and Extensibility.

**Notation.** A Notation or concrete syntax is a set of signs that enable to represent models. A modeling language may have two types of notation: graphical/visual or textual. **Dimensions**: Representation Type and Supporting Mechanisms.

**Size.** Completeness is one of the biggest challenges regarding the development of modeling languages and respective models [Baader et al. 2003, Hoppenbrouwers et al. 2005, Teeuw and van der Berg 1997]. This property states that the language’s meta-model shall include the most important concepts. Therefore, the meta-modeling approach is essential for assuring that the modeling languages allow producing concrete quality models. **Metrics**: Number of Views, Number of Classifiers and Number of Relationships.

**Usability.** The language must be easy for the team members to use, so that the connection between these and computing platforms can be established naturally as if it was an interface [Barišić et al. 2012]. It must also specify system requirements, structures and behaviour. The processes shall be easy to model, the language environment should offer pre-defined constructs and libraries of high-level concepts, these should be easy to adapt to individual needs and the modeling methods need to be comprehensible and well documented [Teeuw and van der Berg 1997]. **Dimensions**: Understandability, Learnability, Operability, Attractiveness, User Satisfaction [ISO/IEC 9126-1 2001] and Adaptability [Teeuw and van der Berg 1997].

**Other Features.** It is a set of MLs’ properties that provide additional information about them. They may or may not be unique abilities. They will be considered, but not weighted to calculate the MLs’ quality values.

### 4.1.2 Specific Properties—UIMLs

**Application Actions.** This property lists a series of actions that the generated application can support, in different contexts.

**User Interactions.** This property intends to show which ways users can interact with the application, whether it is with mouse, keyboard, touch or other means.

**Widget Types.** This property lists a set of graphical user interface elements (either structural or behavioural) that the language makes available for the designer.

### 4.1.3 Specific Properties—BPMLs

**Executability.** A language is executable as long as the user can produce behavioural models with enough fine granularity so they can run as programs. In order to this translation happen effectively, the compiler must be fast and reliable and the generated code must also be fast and robust [Rumpe 2014]. There have been model execution tools and environments for years, but the scalability is always a challenge. Each tool defined its own semantics for model execution, often including a proprietary action language, and models developed in one tool could not be interchanged with or interoperate with models developed in another tool2.

**List of Actions.** This property intends to list all the actions included in business processes that are possible to represent with the modeling language.

---

Modularity. It can be defined as the capacity of construct and re-construct some parts of the models, in a way that is easy for any user to collaborate and accept them. Modules can be related 1-to-1 to interfaces, so that model's structure becomes more visible [Maxwell and Costanza 1997].

4.2 ARENA Website

Apart from useful, we wanted ARENA to be available to as many modelers and developers as possible. This tool intends to be a support for quality choice destined to academic researchers as well as enterprise professionals, at a world-wide level. Therefore, we have developed the ARENA website at http://arenaframework.comeze.com, with PHP and minimal CSS. The framework’s homepage has a brief description of the tool’s goal and context, the list of languages that a user can compare and the possibility of editing a listed language or add a new one.

This tool is being updated regarding language's data and some features are being developed. We want to group languages according to its domain, namely UIMLs, BPMLs or GPMLs and add the possibility of selecting the domain’s languages, instead of one by one, at the homepage. Also, regarding countable items, we believe that adding charts will make information analysis smarter, as well as adding links to language's compatible software, papers and projects in which they were used.

5. VALIDATION AND RESULTS

This section presents the results obtained for the seven analyzed modeling languages, as well as a brief discussion of those results, taking into account its specific domains.

5.1 Evaluation

Table 2 summarizes the main properties of four analyzed UIMLs — UMLi [Silva 2002], UsiXML [Limbourg et al. 2004], XIS [Martins and Silva 2007] and XIS-Mobile [Ribeiro 2014]. The lines Number of Views, Number of Classes and Number of Relationships represent the metrics with the same name, from ARENA's property Size. The ARENA property Notation is represented by two dimensions: Representation Type and Supporting Mechanisms. The line Compatibility refers to a dimension from ARENA's property Interoperability, as well as metrics Number of Compatible Applications and Number of Integration Mechanisms. ML's Functionality is evaluated using dimensions Pattern Usage, Tool Support - M2M Transformations, Tool Support - M2T Transformations and Tool Support - Validation. The last three lines have exactly the same name and meaning of the specific properties, as have shown on the previous chapter, subsubsection 4.1.2.

In a similar way, Table 3 enframes the principal characteristics of the three BPMLs we chose — UML [Object Management Group 2011], BPMN [Object Management Group 2013] and DEMO [Dietz 2006]. The only difference regarding Table 2 is that for this domain, the specific properties are Executability, List of Actions and Modularity, as we presented on subsubsection 4.1.3.
## Table 2. UIMLs comparison based on the ARENA Framework

<table>
<thead>
<tr>
<th>Property Group</th>
<th>Language</th>
<th>Property</th>
<th>UML</th>
<th>UniXML</th>
<th>JSIS</th>
<th>XIS-Mobile</th>
<th>Average [UIMLs]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Views</td>
<td>Number of Views</td>
<td>Abstract Syntax</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>Number of Classifiers</td>
<td>Number of Classifiers</td>
<td>Abstract Syntax</td>
<td>5</td>
<td>12</td>
<td>18</td>
<td>46</td>
<td>22</td>
</tr>
<tr>
<td>Number of Relationships</td>
<td>Number of Relationships</td>
<td>Abstract Syntax</td>
<td>11</td>
<td>20</td>
<td>12</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>Representation Type</td>
<td>Supporting Mechanisms</td>
<td>Concrete Syntax</td>
<td>Graphical</td>
<td>Textual</td>
<td>Graphical</td>
<td>Graphical</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Core metamodel</td>
<td>Eclipse; Enterprise Architect, Microsoft Visual Studio, Notepad++, Kate, WtSchools.com and others</td>
<td>UML Profile</td>
<td>UML Profile</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>Compatibility</td>
<td>Compatibility</td>
<td>ArgoUML</td>
<td>Composites, Single Choice, Multiple Choice, List Selection, Continuous Filter, Menu Navigation, Grid Layout, Tab Menu, Tabular Set, Double List</td>
<td>Composites, Single Choice, Multiple Choice, List Selection, Single Text Entry, Multiple Text Entry, Springboard, List Menu, Tab Menu, Option Menu and 10 more</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pattern Usage</td>
<td>Pattern Usage</td>
<td>General Properties</td>
<td>Abstract Presentation Pattern, Concrete Interaction Object</td>
<td>Concrete &amp; Task Model, Abstract UI, Concrete UI, Final UI, Inter-model mapping, Context translation</td>
<td>Composites, Single Choice, Multiple Choice, List Selection, Continuous Filter, Menu Navigation, Grid Layout, Tab Menu, Tabular Set, Double List</td>
<td>Composites, Single Choice, Multiple Choice, List Selection, Single Text Entry, Multiple Text Entry, Springboard, List Menu, Tab Menu, Option Menu and 10 more</td>
<td></td>
</tr>
<tr>
<td>Tool Support - MIM Transformations</td>
<td>Tool Support - MIM Transformations</td>
<td>No</td>
<td>(N/A)</td>
<td>No</td>
<td>Yes, using Enterprise Architect’s MUG technologies</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tool Support - M2T Transformations</td>
<td>Tool Support - M2T Transformations</td>
<td>No</td>
<td>(N/A)</td>
<td>Yes, defining architectures, templates, and interface generation processes that are compatible with Windows Forms, .NET and ASP.NET platforms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tool Support - Validation</td>
<td>Tool Support - Validation</td>
<td>ARGO validates both UML and UML models, due to both grammars are specified in terms of the UML metamodel</td>
<td>(N/A)</td>
<td>Eclipse .NET Framework and Project T’s three components: Requirements, UML Modeler and MDD Code Generator</td>
<td>XIS-Mobile Framework, namely EA’S Model Validator, supports model validation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Compatible Applications</td>
<td>Number of Compatible Applications</td>
<td>1</td>
<td>&gt;6</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Number of Integration Mechanisms</td>
<td>Number of Integration Mechanisms</td>
<td>3 (ARGO, OCL rules and LOTOS rules)</td>
<td>4 (MDX Technologies, XML Parser, UML2.NET and UML2Distri)</td>
<td>2 (Eclipse .NET and UML Framework)</td>
<td>2 (MDX Technologies, XML Parser)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other Features</td>
<td>Other Features</td>
<td>It is possible to model Activity Diagrams in UMLI, using the UML Case Diagram and the Initialisation construct.</td>
<td>It implements the p7 concept, as it is device-, user-, culturally-, organization-, context-, model- and platform-independent.</td>
<td>Supports Windows Forms, .NET, ASP.NET and WPF, using Model-to-View transformations.</td>
<td>Model validation using a set of rules defined in C# implemented with EA’s Automation Interface. It is possible to generate User-Interfaces for models in EA.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application Actions</td>
<td>Application Actions</td>
<td>Specific Properties</td>
<td>(N/A)</td>
<td>CRUD, OK, Cancel, Navigate, Select, Choose, Associate, other customizable actions</td>
<td>CRUD, OK, Cancel, Navigate, Select, Choose, Associate, other customizable actions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Interactions</td>
<td>User Interactions</td>
<td>Click, Select, (Keyboard) Type, Scroll</td>
<td>Push button, List bar, Check box, Window, Panel, Tab, Cell, Dialog box, Embedded multimedia, Menu, Spin button</td>
<td>Button, Text box, List, Menu, Window, Link, Search bar, Checkbox, Radius button, Drop-down list, Form, Dialog, Label and 15 more</td>
<td>Button, Text box, List, Menu, Window, Link, Search bar, Checkbox, Radius button, Drop-down list, Label, Image, Date Picker and 9 more</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Widget Types</td>
<td>Widget Types</td>
<td>Label, Text field, Combobox, Selectable list, Button</td>
<td>(N/A)</td>
<td>(N/A)</td>
<td>(N/A)</td>
<td>(N/A)</td>
<td>(N/A)</td>
</tr>
</tbody>
</table>

Table 2. UIMLs comparison based on the ARENA Framework.
<table>
<thead>
<tr>
<th>Property</th>
<th>Property Group</th>
<th>UML. (Activity Diagram)</th>
<th>BPML Business Process Diagram</th>
<th>DEOMO (Process Model)</th>
<th>Average (BPMLs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Views</td>
<td>Abstract Syntax</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Number of Classes</td>
<td>Abstract Syntax</td>
<td>43</td>
<td>19</td>
<td>6</td>
<td>26</td>
</tr>
<tr>
<td>Number of Relationships</td>
<td>Abstract Syntax</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Representation Type</td>
<td>Concrete Syntax</td>
<td>Graphical</td>
<td>Graphical</td>
<td>Graphical</td>
<td>—</td>
</tr>
<tr>
<td>Supporting Mechanisms</td>
<td>MDIF</td>
<td>MDIF</td>
<td>MDIF</td>
<td>MDIF</td>
<td>—</td>
</tr>
<tr>
<td>Pattern Usage</td>
<td>General Properties</td>
<td>30</td>
<td>86</td>
<td>1</td>
<td>46</td>
</tr>
<tr>
<td>Tool Support - R2M Transformations</td>
<td>Yes, it is possible to transform UML and Activity, Activity Start, and Business Process diagrams into BPMN, using UML2.0 Diagram Interchange.</td>
<td>Yes, BPMN's Diagram (interaction) definition provides a basis for modeling and interchanging graphical notations, specifically for activities in BPMN, UML and SysML.</td>
<td>No, it is only partially defined, but it is not implemented.</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>Tool Support - XMI Transformations</td>
<td>Yes, IBM Rational Software Architect allows UML to Java transformations. Also, it is possible to transform UML models into communicating sequential processes (CSP) using graphs.</td>
<td>Yes. It can be mapped to WS-BPEL and XML, and possibility to Finite State Machines.</td>
<td>All compatible tools perform syntax analysis on DEMO models and return &quot;warning&quot; or &quot;OK&quot; messages.</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>Tool Support - Validation</td>
<td>XML uses XSD to validate instances of XML documents. The &quot;implementation&quot; attribute, which is present in UML activity, Activity Start, and Business Process Diagrams.</td>
<td>XML uses XSD to validate instances of XML documents. The &quot;implementation&quot; attribute, which is present in UML activity, Activity Start, and Business Process Diagrams.</td>
<td>XML uses XSD to validate instances of XML documents. The &quot;implementation&quot; attribute, which is present in UML activity, Activity Start, and Business Process Diagrams.</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>Number of Compatible Applications</td>
<td>41</td>
<td>59</td>
<td>5</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>Number of Integration Mechanisms</td>
<td>UML-driven and Extensible. Generates languages using reverse-engineering. Can be integrated with IDEs. Can be integrated with Web Browsers and can be used with Office applications.</td>
<td>UML-driven and Extensible. Generates languages using reverse-engineering. Can be integrated with IDEs. Can be integrated with Web Browsers and can be used with Office applications.</td>
<td>UML-driven and Extensible. Generates languages using reverse-engineering. Can be integrated with IDEs. Can be integrated with Web Browsers and can be used with Office applications.</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>Other Features</td>
<td>UML diagrams allow interchanging between XSD and XMI. Also, BPMN has Diagram Interchange package, a set of BPMN meta classes that allows BPMN models' interoperability between different tools.</td>
<td>It is possible to add comments on models.</td>
<td>It is possible to add comments on models.</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>A first-class, multiplatform library provided that is used to define the UML metamodel, as well as other architecturally related metamodels, such as MOF or OMG.</td>
<td>It is possible to copy data between graphical elements, using the same itemDefinition or a DataAssociation with a transformation Expression</td>
<td>It is possible to copy data between graphical elements, using the same itemDefinition or a DataAssociation with a transformation Expression</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>Extensibility</td>
<td>UML Profiles can be used to customize the language for particular platforms and domains.</td>
<td>It has an Extension Class that allows extending standard BPMN elements with additional attributes.</td>
<td>Extensions have been added to this way, modeling an organization's activities with DEMO and then extending them to allow activities to be executed using different systems.</td>
<td>—</td>
<td></td>
</tr>
<tr>
<td>Executability</td>
<td>Specific properties</td>
<td>There are specific types of elements that can be deployed as Executable Artifacts and can be related to a node using the DeploymentArtifact and DeploymentTarget elements.</td>
<td>There are specific types of elements that can be deployed as Executable Artifacts and can be related to a node using the DeploymentArtifact and DeploymentTarget elements.</td>
<td>—</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. BPMLs comparison based on the ARENA Framework
The quality value of a language is determined by Formula 1, designated as the ARENA General Equation of Quality. The names of the plots to be firstly multiplied and then added, are no more than shortened versions of the properties considered in Tables 2 and 3.

\[
\text{QualityML}_n = \sum_{i=1}^{j} w_i \cdot r_i
\]

\[
= w_{\text{absStxProps}} \cdot r_{\text{absStxProps}} + w_{\text{concStxProps}} \cdot r_{\text{concStxProps}} +
+ w_{\text{genProps}} \cdot r_{\text{genProps}}
\]

\[
= w_{\text{numViews}} \cdot r_{\text{numViews}} + w_{\text{numClassifs}} \cdot r_{\text{numClassifs}} +
+ w_{\text{numRelats}} \cdot r_{\text{numRelats}} + w_{\text{repType}} \cdot r_{\text{repType}} +
+ w_{\text{suppMechs}} \cdot r_{\text{suppMechs}} + w_{\text{numPatterns}} \cdot r_{\text{numPatterns}} +
+ w_{\text{M2MTrfs}} \cdot r_{\text{M2MTrfs}} + w_{\text{M2TTTrfs}} \cdot r_{\text{M2TTTrfs}} +
+ w_{\text{Valid}} \cdot r_{\text{Valid}} + w_{\text{numCompApps}} \cdot r_{\text{numCompApps}} +
+ w_{\text{numIntMechs}} \cdot r_{\text{numIntMechs}}
\]

(1)

Since we are comparing two different specific domains — User-Interface and Business Process —, we need to create two specific equations based on ARENA General Equation of Quality, as displayed on Formulas 2 and 3.

\[
\text{QualityUIML}_n = \text{QualityML}_n + \sum_{i=1}^{j} w_{\text{SpecUIProps}_i} \cdot r_{\text{SpecUIProps}_i}
\]

\[
= \text{QualityML}_n + w_{\text{numAppActs}} \cdot r_{\text{numAppActs}} +
+ w_{\text{numUserInts}} \cdot r_{\text{numUserInts}} + w_{\text{numWidgets}} \cdot r_{\text{numWidgets}}
\]

(2)

\[
\text{QualityBPML}_n = \text{QualityML}_n + \sum_{i=1}^{j} w_{\text{SpecBPProps}_i} \cdot r_{\text{SpecBPProps}_i}
\]

\[
= \text{QualityML}_n + w_{\text{numActions}} \cdot r_{\text{numActions}}
\]

(3)

The notation used on the formulas above has the following meaning: \(i\) — Framework’s metric/dimension; \(j\) — Sum of the number of metrics with the number of dimensions; \(n\) — Language’s
name; \( r \) — Metric's/dimension's rate; \( w \) — Metric's/dimension's weight. The rating each property can have is the following: 1 — Very Low; 2 — Low; 3 — Medium; 4 — High; 5 — Very High. If quantifiable and with equal level of importance, each property’s value is compared towards that property’s average value in this work. Its rating is given according to how far it is from the average, upwards or downwards, according to the following ranges: 
\[
Value < \text{AvgValue} - 1/2 * \text{AvgValue} \quad \text{or} \quad Value \in [\text{AvgValue} - 1/2 * \text{AvgValue}, \text{AvgValue} - 1/4 * \text{AvgValue}] \quad \text{or} \quad Value \in [\text{AvgValue} - 1/4 * \text{AvgValue}, \text{AvgValue} + 1/4 * \text{AvgValue}] \quad \text{or} \quad Value \in [\text{AvgValue} + 1/4 * \text{AvgValue}, \text{AvgValue} + 1/2 * \text{AvgValue}] \quad \text{or} \quad Value > \text{AvgValue} + 1/2 * \text{AvgValue}.
\]
The exceptions (non-quantifiable properties) are: Representation Type — The most popular value has higher rating than the least; Supporting Mechanisms — Since it's very specific for each language, this rating reflects easiness to understand and extend the metamodel; Compatibility — It may be unique for each language, so its rating is focused on the programs’ general usability and functionality; and Tool Support (Validation, Model to Model and Model to Text) — Since the programs referred on the Compatibility property generate models, we assumed that Validation is correctly done, as well as the transformations.

The languages in which we don’t know how these characteristics are covered were assigned 1 (Very Low) and the languages that don’t have these features were assigned 2 (Low). Each rated property will be multiplied by its weight, as referred on section 4.1, and the sum of this products will return the language’s quality, according to the ARENA General Equation of Quality.

5.2 Discussion

5.2.1 UIMLs

In a related paper [Morais and Silva 2015], we also have compared these four UIMLs. It is possible to see that, for the same domain, the languages do not differ much in terms of number of views and number of relationships, but they do when it comes about supporting mechanisms and Compatibility. This can be explained by their maturity and popularity. Also, there are patterns concerning number of classifiers and notation, both with a single exception.

If the Number of Views seems to be balanced for these four languages (all have between 4 and 6), it is not possible to conclude the same about the Number of Classifiers, because UsiXML, UMLi and XIS have, respectively, 12, 13 and 18 and XIS-Mobile has 46, resulting in a difference of 34 items to the lowest number. Less unbalanced is the Number of Relationships — it varies from UMLi’s 11 to UsiXML’s 20.

For Representation Type, clearly there is a preference for Graphical in disfavour of Textual, probably due to easiness to see the models in a clearer way and according to the WYSIWYG paradigm. About Supporting Mechanisms, as expected due to their nature, XIS and XIS-Mobile have the same value, opposing to UMLi that is an extension to UML and therefore uses a simpler approach and UsiXML that, being based on XML, results in a different architecture that is complemented by Cameleon. Considering Compatibility, all languages follow different tendencies, with a clear advantage to UsiXML. This might be due to XML’s universality (as it is used in RSS, SOAP messages and XHTML), it is both human- and machine-readable and it is capable of representing all Unicode characters.

XIS is the top language concerning to Application Actions, since it allows the software to do at least 9 different actions. Contrasting with that, UsiXML is the better one in terms of User Interactions, since it provides 12 gestures that can be done, while XIS and UMLi only have 4 defined. The language that includes more Patterns is XIS-Mobile, Composite being the only one of Structural Design type and the remaining 19 of UI Design, that is 10 more than XIS. Still about this property, UsiXML has 4 which are based on Cameleon reference framework and UMLi has 2 patterns, each for a UML package that was extended.
About Tool Support - Validation, each language has its own method, always influenced by the compatible applications and the ones shown in Compatibility line, which in many cases, are the same. We don’t know how it’s done for UsiXML, but for XIS it is the most extensive process. Now referring to Tool Support - Model to Model Transformations, only XIS-Mobile has this feature implemented, taking advantage of EA’s capabilities, while regarding Tool Support - Model to Text Transformations, XIS and XIS-Mobile provide this MDD-core feature, as opposed to UMLi, with a great focus on Web technologies. Again, we know nothing about these two properties on the UsiXML case.

<table>
<thead>
<tr>
<th>Property</th>
<th>Language</th>
<th>Weight</th>
<th>UMLi</th>
<th>UsiXML</th>
<th>XIS</th>
<th>XIS-Mobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Views</td>
<td></td>
<td>0.033</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Number of Classifiers</td>
<td></td>
<td>0.033</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Number of Relationships</td>
<td></td>
<td>0.033</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Representation Type</td>
<td></td>
<td>0.033</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Supporting Mechanisms</td>
<td></td>
<td>0.033</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Compatibility</td>
<td></td>
<td>0.033</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Pattern Usage</td>
<td></td>
<td>0.1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Tool Support - M2M Transformations</td>
<td></td>
<td>0.1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Tool Support - M2T Transformations</td>
<td></td>
<td>0.1</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Tool Support - Validation</td>
<td></td>
<td>0.1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Number of Compatible Applications</td>
<td></td>
<td>0.05</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Number of Integration Mechanisms</td>
<td></td>
<td>0.05</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Application Actions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Interactions</td>
<td></td>
<td>0.1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Widget Types</td>
<td></td>
<td>0.1</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Total Quality</td>
<td></td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4. UIMLs evaluated with ARENA Framework

Now regarding the Number of Compatible Applications, all languages have the same value, therefore we considered than none stood out more than the other. It seems to be a bit contradictory to the Compatibility property. The difference is that the latter doesn’t take into account the language’s suitable program but the produced models’ compatibility with other platforms. For the Number of Integration Mechanisms, despite some are easier to work with (XML Parser) than others (LOTOS rules), all languages have received the same rating, due to the number is close to the average of this property, in this work.

The attractive and useful Widget Types inform that XIS and XIS-Mobile have a higher number of items (28 and 22, accordingly), although not exactly the same. UsiXML stands with 11 and UMLi has only 5, which is expected from an extension that claims to “be a conservative extension of UML” and "should introduce as few new models and constructs into the UML as possible” [Silva 2002].

According to Table 4, for this evaluation, XISMobile has the highest quality value—4—among these four UIMLs, in the considered domain, as we concluded previously [Morais and Silva 2015].

5.2.2 BPMLs
In this subsubsection, we present Table 5, in which the considered three BPMLs’ most relevant properties are evaluated by our framework. At a glance, it is possible to see that generally, UML and BPMN are more developed than DEMO. This difference is clear regarding the number of classifiers, of relationships, of patterns, of compatible applications and the transformation possibilities.
Since we considered only diagrams focused on actions and process specification, it is plausible a low Number of Views: both UML and BPMN only have one, the diagram itself, yet DEMO has two (Process Structure Diagram and Transaction Process Diagram). Respecting Number of Classifiers and Number of Relationships, DEMO has a smaller value than the other two languages. Albeit there is a big distance in terms of UML’s 43 and BPMN’s 29 classifiers to DEMO’s 6, it doesn’t mean that the latter is incomplete. It may not provide as much flexibility as the other two, but surely it is simpler to understand and has less redundancy probability. The difference between them is reduced when comparing the Number of Relationships, as they all are close to the average value, 5.

Referring now to Representation Type, there is an unanimity. All selected languages are Graphical, which allow process specification and hierarchy to be easier than if only text was used, taking into account the number of concepts and graphical elements that this domain contains. This choice is in line with the MDSD approach and its advantages, as we presented in Section 2.5. Considering Supporting Mechanisms, UML and BPMN have a MOF meta-model and DEMO has an extended BNF grammar. The reason for this difference is that when developing DEMO, Jan Dietz and the EE team have chosen a non-object-oriented approach, because they believe it would be too restrictive, so they preferred to define grammars as they are more open to extensibility and compatible with other languages. Regarding Compatibility, the two OMG’s languages share some common IBM, Microsoft and Sparx Systems applications, while the third has Microsoft Windows’ compatible tools and web-based tools for Apple and Linux users.

For Tool Support - Model to Model Transformations and Model to Text Transformations, UML and BPMN have a clear advantage over DEMO, due to OMG’s development focus on interchangeable diagrams, namely between themselves, as they both have a Diagram Interchange package, which allows node-to-node mapping and their models can be exported to two well-known and wide-used text languages, Java and XML. DEMO has no available transformations, either because it is still planned to be developed or because they do not see advantages on them. As for Tool Support - Validation, the languages have different approaches: both UML and BPMN use XSD to validate XML documents, but BPMN has specific validation mechanisms for some elements’ attributes and DEMO only has a syntax analysis algorithm, since its models are very simple, due to the smaller number of graphical elements.

<table>
<thead>
<tr>
<th>Property</th>
<th>Language</th>
<th>Weight</th>
<th>UML (Activity Diagram)</th>
<th>BPMN (Business Process Diagram)</th>
<th>DEMO (Process Model)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Views</td>
<td>0.933</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Number of Classifiers</td>
<td>0.933</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Number of Relationships</td>
<td>0.933</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Representation Type</td>
<td>0.933</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Supporting Mechanisms</td>
<td>0.933</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Compatibility</td>
<td>0.933</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Pattern Usage</td>
<td>0.1</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Tool Support - M2M Transformations</td>
<td>0.1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Tool Support - M2T Transformations</td>
<td>0.1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Tool Support - Validation</td>
<td>0.1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Number of Compatible Applications</td>
<td>0.05</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Number of Integration Mechanisms</td>
<td>0.05</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Other Features</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Reusability</td>
<td>0.1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Extendibility</td>
<td>0.1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Executability</td>
<td>0.1</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Total Quality</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Table 5. BPMLs evaluated with ARENA Framework
About the Number of Compatible Applications, BPMN leads with an impressive number of 58, followed by UML's 41 and, above the average, comes DEMO with 5. Since DEMO is the most recent one and, therefore, the most unknown, this difference can be explained. However, we believe that when more universities and companies adopt it and see its value on its simplicity, more tools will be developed and more accepted this language will be. The BPMLs are well balanced in terms of Number of Integration Mechanisms, with an average 4, the same as DEMO's, higher than BPMN's 2 and lower than UML's 7.

BPMN is the top language in what comes about Pattern Usage. It has 86, which are divided in Control-Flow (28), Data (18), Resource (8) and Exception Handling (32). DEMO only has 1, the Transaction (connection between two agents, which is composed by 20 steps) and UML has 25 Control-Flow patterns, 17 Data patterns and 8 Resource patterns, making a total of 50. As far as Reusability is concerned, UML contains a library which allows editing its metamodel or others, such as Meta Object Facility (MOF) or CommonWarehouse Metamodel (CWM), while BPMN has a more model-focus approach, making it possible to copy data between elements. We can't compare DEMO in this topic, since we could not find related information. In terms of Extensibility, the UML can be extended in two ways: A new dialect of UML can be defined by using Profiles to customize the language for particular platforms and domains, or a new language related to UML can be specified by reusing part of the Infrastructure-Library package and augmenting with appropriate metaclasses and metarelationships. BPMN includes an Extension Class composed by four elements, that allows extending standard BPMN elements with additional attributes, such as Artifacts. Still, they need to have valid BPMN Core, be semantically compatible with any BPMN element and extended Diagrams should keep the basic look-and-feel to maintain easy understanding, which means that Events, Activities and Gateways must not be altered [Object Management Group 2013]. EE has been doing an effort in this way with DEMO, taking already developed models or modeling an organization's activities with this language and then extending them to allow activities' generation, operation and discontinuation.

And lastly, about Executability and once again, we don't know how this property can be evaluated in the DEMO case, but we realized that UML and BPMN are executable modeling languages. The first one has the possibility of making some components executable, by defining stereotype "executable" in an Artifact and then relating them to nodes, using deployment linking elements. The latter has some restrictions implemented (as listed on the bottom line of Table 5) so the executable models can be emitted and its private business process models must have the "isExecutable" boolean set to "TRUE", so they can become executable.

From our point of view, we defend that for the considered domain and the languages selected, BPMN has the highest quality value — 4 — compared to the other two BPMLs.

6. CONCLUSION

This is the last section of this extended abstract. It presents the final remarks of this work, that is, the conclusions we achieved and some directions of future work that this thesis can point to.

This work presents an overview of academic research related to the problematic of choosing the best from several modeling languages and the quality assessment frameworks as a response to that problem. We have evaluated four UIMLs and three BPMLs using ARENA, a framework oriented to that solution. In this case, ARENA's most adequate dimensions and metrics were used, along with User-Interface and Business Process specific properties, defined for this purpose and context. With this analysis, it is possible to list the features in which each UIML and BPML stands out from the others, either because they are better or they are unique. Clearly there is a trend for using graphical
notations. This can be due to guidelines that compose the well-known Model-Driven Engineering. An advantage, is also to use these graphical models to implement the alternative approach to produce software: Model-Driven Software Development.

With this report, one can conclude that MDSD seems to be the optimal approach to future code generation but there is still a lot of work to do. MDSD has some issues to deal with, such as: The rigidity of modeling (i.e., there’s no possibility of changing every detail); The flexibility depends on the selected tool and DSL; The roles of project members are rather different (e.g., the person who builds the solution is the Business Engineer, and not the Programmer); The modeling environment doesn’t always support version control; The team normally lose their focus, experimenting new features instead of working on the projects’ objectives and more. Although there are many Domains to even more Modeling Languages, quality is a factor that is important during the development of prototypes, but isn’t still well considered during the selection process. This report will help Software Language Engineers to have an easier modeling language and model brainstorm, in order to help MDSD becoming the next big revolution on Software development.

The chosen metamodel types vary very much from modeling language to modeling language, since some use MOF, others BNF or UML Profiles, and even MVC is still used. Another possible choice is to extend other ML’s metamodel. Tool support is very important, in terms of usability and functionality. They are also responsible to render and validate the produced models, so this aspect can be the quality bottleneck. In terms of look-and-feel, it is very important for the designer to have available design patterns and builder tools, so he can produce an attractive and easy interface. He must also be able to choose between several actions and widgets, so the mobile, web or desktop software application can have an attractive layout and great impact.

Since the foundations of this framework were constructed with this work, we have considered some directions and suggestions of future work that this thesis can point to, in order to turn ARENA into a more useful and world-wide used tool.

Analyze other specific Domains (for instance Software Process Modeling Languages) and other DSMLs, such as Petri Nets (for Mathematical Models) or CML-ML (for CMS-based applications), to provide more credibility and variety to ARENA framework, similarly to the CMS Matrix project, available at http://www.cmsmatrix.org/.

Promote and search workshops related to these and other UIMLs and BPMLs, in order to get a more objective and solid opinion on them, both with people who have worked with MLs as with others that have not worked. XIS-Mobile’s workshop was profitable and if there were more, more easily properties and metrics would be defined and supported. The questions of the related feedback inquiries should be focused on the properties, metrics and dimensions of ARENA.

Apply ARENA to more GPMLs, for instance SysML [Object Management Group 2012], to understand if it is a suitable framework, namely if it has a good balance between generic and specific properties, adding, editing or removing properties and metrics, or if GPMLs must have their own framework.

Study the properties and metrics’ adequacy, according to these new analyses by introducing or removing them and changing weights in the formula that gives the Quality percentage.

---

³ 8 reasons why Model-Driven Development is dangerous*, by Johan den Haan
REFERENCES


Defining domain-specific modeling languages: Collected experiences. In OOPSLA 4th Workshop on Domain-Specific Modeling. ACM.


When and how to develop domain-specific languages. ACM Computing Surveys, 37:316–344.


Assessing business processing modeling languages using a generic quality framework. Norwegian University of Science and Technology.


Xis-mobile: A dsl for mobile applications. In Proceedings of ACM SAC 2014 Conference. ACM.


