Comparative Analysis of Workbenches to Support DSMLs: Discussion with Non-Trivial Model-Driven Development Needs

André Ribeiro[1], Luís de Sousa[2] and Alberto Rodrigues da Silva[1]
[1]*INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal*
[2]*Luxembourg Institute of Science and Technology, Belvaux, Luxembourg*

Keywords: Domain Specific Language, Model-Driven Development, Modeling Tool.

Abstract: The development and use of Domain Specific Languages emerged as a way to cope with complex problems using concepts closer to the problem domain. By leveraging the principles proposed by Model-Driven Development (MDD), like the separation of concerns and the use of model transformations, this approach became popular and caused the emergence of a variety of languages, known as Domain Specific Modeling Languages (DSMLs). Moreover, the use of DSMLs with graphical notations abstracts even more the problem domain, either by using extensions of UML or directly using metamodeling languages. The definition and use of DSMLs is only possible through specific tools, known as languages workbenches. This paper discusses the analysis and comparison of three of these tools (namely Papyrus, Enterprise Architect and Sirius) that were used to create the XIS-Mobile language, a non-trivial DSML defined as a UML profile for modeling mobile applications in a platform-independent way. These tools were evaluated considering a set of key criteria (namely learnability, usability, graphical completeness, validation support, transformation support, evolvability and interoperability) which show their suitability to develop non-trivial languages.

1 INTRODUCTION

The complexity of software systems has increased over the years, especially due to the growing difficulty and specificity of the problems they aim to solve. In order to deal with this issue, several Software Engineering approaches have emerged. Essentially, these approaches seek to raise the level of abstraction at which software systems are developed (Bourguignon, 1990). One of the most popular is the Model-Driven Development (MDD or Model-Driven Engineering) where domain models are able to express concepts specific to a certain domain problem, unlike the third generation programming languages. MDD advocates the use of domain models has first-class citizen entities guiding the development process through techniques like metamodeling, model transformations and code generation. MDD also allows that other artifacts such as source code or documentation could be generated automatically from models (Schmidt, 2006; da Silva, 2015).

Domain models are often expressed using Domain Specific Languages (DSLs). A DSL is a language tailored for a specific set of tasks which, through appropriate notations and abstractions, either textual or graphical, depicts concepts of a particular problem domain (van Deursen, et al., 2000). Particularly, Domain Specific Modeling Languages (DSMLs) abstract even more the problem domain since their concepts are commonly represented using a graphical notation. Since there is not an established convention, whenever we mention DSML, we mean a "DSL with a graphical notation".

Over the last decades some tools were created to support both MDD principles and DSLs. The earlier ones put forth in the 1980s and 1990s, were the Computer-Aided Software Engineering (CASE) tools. CASE tools focused on providing developers with the tools and methods to express software systems through general-purpose language representations. Despite the attention they attracted, CASE tools failed to be widely adopted and had little impact on commercial software development chiefly due to: (1) the poorly mapping of general-purpose language representations onto the underlying platforms, which caused the generation of large amounts of code, harder to understand and maintain; and (2) the inability to scale up to handle complex systems, since the tools did not support

Ribeiro, A., Sousa, L. and Silva, A.
Comparative Analysis of Workbenches to Support DSMLs: Discussion with Non-Trivial Model-Driven Development Needs.
DOI: 10.5220/0005745603230330
In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, pages 323-330
ISBN: 978-989-758-168-7

concurrent engineering, lacked integration of the generated code with other platforms and their graphical representations were too generic and non-customizable to be applied on several domains (Schmidt, 2006). Thereafter, both the evolution of programming languages and the lessons learned from CASE tools caused the emergence of increasingly better MDD and DSL supporting tools. More recently, language workbenches are popular solutions to develop DSLs and to overcome the inflexibility of the CASE tools. A language workbench is essentially a tool that supports the definition, reuse and composition of DSLs, as well as the creation of customized Integrated Development Environments (IDEs) (Fowler, 2005). In essence, a language workbench allows not only the creation of DSLs, but also the creation of custom IDEs for their usage. For that reason, language workbenches are also known as meta-CASE tools (Costagliola, et al., 2006).

During the last years our research group has conducted several works in the area of MDD, namely developing DSMLs defined as UML profiles, such as (de Sousa Saraiva & da Silva, 2009) (Ribeiro & da Silva, 2014) (de Sousa & Silva, 2015) (Filipe, et al., 2016). In this paper we provide an analysis and discussion of this kind of tools based on our most recent experiences. More specifically, we focus on analyzing workbenches when the goal is to define and use a complex UML profile to implement a MDD process. We consider as a complex UML profile, a non-trivial language that contains several concepts and representations (diagrams or views). For that purpose, we have defined an evaluation framework composed of six criteria to describe and analyze the main features of three representative language workbenches (Papyrus, Enterprise Architect and Sirius) using the XIS-Mobile language as case study. The XIS-Mobile language (Ribeiro & da Silva, 2014) is a UML profile that allows the specification of mobile applications in a platform-independent way and is composed of several concepts organized in six views. The main features, limitations and difficulties of implementing a complex DSML, like XIS-Mobile, and its supporting tools are discussed, what provides a relevant contribution to users who also need to implement such DSMLs.

The outline of this paper is structured as follows. Section 2 presents DSL workbenches not directly considered in this paper. Section 3 defines the analysis context by describing the XIS-Mobile language used as case study and the evaluation criteria. Section 4 presents the main features of the analyzed tools. Section 5 presents and discusses the results of this comparative study. Section 6 discusses the related work. Finally, Section 7 concludes the paper summarizing its key points and referring future work.

2 LANGUAGE WORKBENCHES

As mentioned above, a language workbench (or meta-CASE tool (Costagliola, et al., 2006)) is a software tool that supports the definition, reuse and composition of DSLs with the corresponding creation of customized IDEs, providing features such as model edition, validation and different types of model transformations. This paper focuses on the analysis of only three language workbenches, mainly due to reasons of space restrictions and for better explanation. However, if we wanted to broad the scope (e.g. consider not only UML profiles), other popular tools could be mentioned, namely MPS, Xtext, Spoofax, MetaEdit+ or Microsoft DSL Tools.

The Meta Programming System (MPS) (http://www.jetbrains.com/mps) is an open source language workbench developed by JetBrains for the creation of textual DSLs. MPS provides a projectional editor that supports syntactic forms for text, tables or mathematical symbols, allowing the definition of the language, its editor and code generators. The languages produced with MPS can be standalone languages, extensions or compositions of other languages.

Xtext (https://www.eclipse.org/Xtext) is a framework for developing textual DSLs, provided as an Eclipse plug-in. Xtext allows the definition of a DSL using an EBNF grammar language and from it generates a parser, an AST metamodel in EMF and a full-featured Eclipse text editor plug-in. Xtext also integrates with other tools from the Eclipse Modeling Project.

Spoofax (http://metaborg.org/spoofax) is a workbench for the development of textual DSLs with full-featured Eclipse editor plug-ins. Spoofax uses several meta-DSLs: (1) SDF grammar is used to define the syntax of the DSL and from it basic editor services (e.g. syntax highlighting) are automatically created; (2) NaBL for name binding, helping in services like code completion or reference resolving; and (3) Stratego for transformations like refactoring services (e.g. rename) and code generation. From these specifications, Spoofax generates an Eclipse-based editor to use the defined DSL.

MetaEdit+ (http://www.metacase.com/mep) is a proprietary language workbench developed by MetaCase for creating DSMLs. MetaEdit+ allows defining the language concepts and rules graphically or using forms. Then, it is possible to draw the DSL graphical representation using the Symbol Editor. After that, the MetaEdit+ Code Generator is able to generate any kind of code by using code templates defined in its own generator definition language.

Microsoft DSL Tools (http://goo.gl/ArYdjC) allow the design of DSMLs that are hosted in the Visual Studio environment. DSL Tools provide: (1) a graphical designer for the creation and edition of DSL definition and its toolboxes; (2) a validation engine to assure that the DSL is well-formed and to display errors/warnings in case of problems; and (3) a code generator that receives the DSL model as input and outputs source code, using T4 Text templates. After the definition of the DSL, the DSL project can be built and run in a custom Visual Studio instance.

3 ANALYSIS CONTEXT

Before presenting the analysis and comparison of the selected tools, it is important to describe the DSML used as case study, XIS-Mobile, and the criteria considered to evaluate those tools, as well as justify their choice. The driver of this study was the need for testing the definition of the XIS-Mobile language and creating a custom tool that supports its use, and also to investigate the existing tools that could support such a language. Unlike the surveys presented in section 6 (Related Work), which used small and somehow trivial case studies, the use of a language with so many concepts and views as XIS-Mobile proved interesting to highlight other problems and needs.

3.1 XIS-Mobile Language

The XIS-Mobile language (Ribeiro & da Silva, 2014) is a DSML defined as a UML profile, conceived for designing mobile applications in a platform-independent way and using domain specific concepts. This way both complexity and platform fragmentation problems can be mitigated, resulting in increased productivity. XIS-Mobile uses a multi-view organization composed of six views: Domain, BusinessEntities, UseCases, Architectural, InteractionSpace and NavigationSpace. Additionally, XIS-Mobile specifies roughly forty-six types of elements and sixteen types of relationships. This variety of views and concepts with different meanings make XIS-Mobile a complex and non-trivial language.

The Domain view describes the domain entities relevant to the problem domain, their attributes and the relationships among them. The BusinessEntities view represents higher-level entities, called business entities, which provide context to a certain use case and interaction space. The UseCases view details the operations a user can perform in the context of a business entity and/or an external service. The Architectural view depicts the interactions between the mobile application and other external entities (e.g. internal providers or servers), it may also be considered as a "distributed systems view". The InteractionSpace view is perhaps the most complex view of XIS-Mobile due to the amount of abstractions it contains. This view describes each application's screen, known as interaction space, namely the UI layout, the events a certain UI component can trigger and the gestures that can be performed. The NavigationSpace View describes the navigation flow between the various interaction spaces of the application.

3.2 Evaluation Criteria

This study considers six evaluation criteria used to assess each one of the analyzed tools. These criteria have been derived from our own experience and previous work in this area, as well as from the analysis of related work like (Amyot, et al., 2006) (Saraiva & da Silva, 2008) (Vasudevan & Tratt, 2011) that is discussed in detail in section 6. Thus, we considered the following major criteria:

- **Learnability** – Time and effort required to learn and produce the DSL workbench;
- **Usability** – Ease of creation and manipulation of the DSL elements and companion workbench;
- **Graphical Completeness** – Ability to depict all the DSL elements representation;
- **Validation Support** – Ability to validate the models produced;
- **Transformation Support** – Ability to support Model-to-Model (M2M) and/or Model-to-Text (M2T) transformations;
- **Evolvability** – Ability to continuously support former models when the languages (and their metamodels) evolve;
- **Interoperability** – Ability to export/import the models using a standard format, such as XMI.

4 ANALYZED TOOLS

Throughout this paper there were chosen three modeling tools available online. We chose only three tools for reasons of space restrictions and simplicity. These tools were selected according to the following requirements: (1) the tool should support the creation of graphical DSLs based on UML profiles, since XIS-Mobile is a UML profile; (2) the tool should have good and up-to-date documentation; (3) the tool should be popular and recent, or at least should have an active user community; and (4) the tool should be free or available under an academic license. Therefore, the choice was on Papyrus, Enterprise Architect and Sirius that will be briefly described. It is important to note that other tools could have been chosen instead (e.g. Obeo's UML Designer, Visual Paradigm or Rational Software Architect), or even some of those mentioned in Section 2 if we would consider a broader scope.

4.1 Papyrus

Papyrus is an Eclipse plug-in belonging to the Model Development Tools (MDT) sub-project which provides an integrated environment for the edition of any kind of Eclipse Modeling Framework (EMF) based model, including UML and related languages like SysML and MARTE.

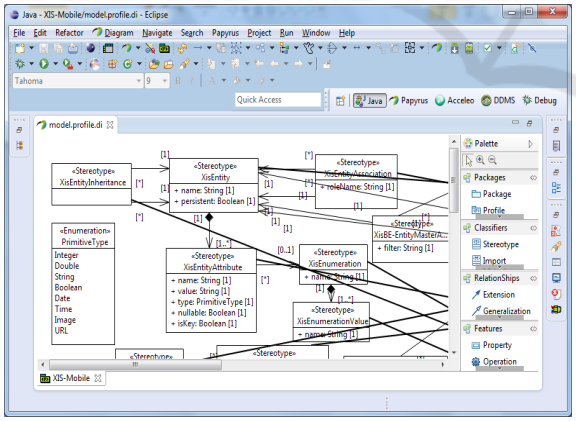

Figure 1: XIS-Mobile metamodel definition using Papyrus.

Papyrus is fully compliant with the OMG specification for UML2 and provides graphical editors for all of its diagrams. Namely, Papyrus offers advanced support for UML profiles allowing users to define custom UML-based DSLs and their editors with custom palettes. Papyrus also allows the integration of these DSL editors with other Eclipse MDD tools that provide, for instance, model validation (e.g. OCL or EVL), model transformation (e.g. ATL or QVT) and code generation (e.g. Acceleo or JET). The definition of models is mainly performed using a graphical editor, but it is also possible using a tree style hierarchy view.

Once completed, all the information that builds up the language workbench can be deployed as a regular Eclipse IDE plug-in. Papyrus is documented with tutorials, examples, videos, wiki and has a dedicated forum. Papyrus is freely available under the Eclipse Public License (EPL). Figure 1 shows the creation of the XIS-Mobile metamodel using Papyrus.

4.2 Enterprise Architect

Enterprise Architect (EA) is a visual modeling tool, developed by Sparx Systems, for the design of software systems, for business process modeling and for more generalized modeling purposes. EA is based on the OMG's UML 2.5 specification supporting the creation of all UML diagrams, elements and connectors. Therefore, EA supports very well the definition of new DSMLs based on UML profiles. This support is powered by the EA's Model Driven Generation (MDG) Technologies that not only allow specifying the UML profile, but also allow extending EA with custom toolboxes, diagrams, project templates and patterns, among other features.

EA allows releasing all of these components customized to a certain DSML as an add-in extending EA's functionality. EA's add-ins can be written in C#, Visual Basic or Java and also provide access to EA's Automation and Validation Interfaces. The Automation Interface allows accessing and manipulating the models, as well as extending EA's User Interface. For instance, the Automation Interface can be a useful resource when the user needs to create new models automatically.

In turn, the Model Validation Interface allows the definition of model rules or constraints to be performed on the model. These rules can have a severity level (Warning or Error) and a custom message associated, and whenever a rule message is clicked the user will navigate to the element that produced the infringement.

EA provides documentation with a user guide, demonstration videos, tutorials and a user forum. EA is a proprietary tool, but has a 30-day trial version. After that, a license is needed with prices depending on the EA version and the type of user. For example,

a license for the Desktop Edition of EA for an individual user is US$135. Figure 2 depicts the creation of the XIS-Mobile metamodel using EA.

4.3 Sirius

Sirius is an Eclipse plug-in that allows creating custom graphical modeling workbenches by leveraging the Eclipse Modeling technologies, namely EMF and GMF. It has been created by Obeo and Thales to provide a generic workbench for model-based architecture engineering that can be easily customized to fit specific needs on specific domains.

Sirius allows the specification of a modeling workbench using graphical, table and tree editors which enable users to create, edit and visualize their EMF-based models. Editors are defined by a model called Viewpoint Specification Model (VSM), which defines the complete structure of the modeling workbench, its behavior and the edition and navigation tools. This model is dynamically interpreted by a runtime within Eclipse and therefore allows users to test the associated modeling workbench instantly. Once completed, the modeling workbench can be deployed as a regular Eclipse IDE plug-in.

Similarly to other Eclipse components, Sirius is documented with tutorials, examples, wiki, videos and has a dedicated user forum. Additionally, Sirius is available for free under the EPL. Figure 3 illustrates the VSM for XIS-Mobile in Sirius.

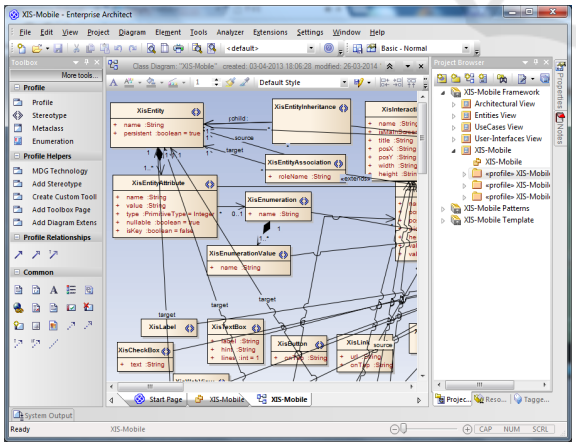

Figure 2: XIS-Mobile metamodel definition using EA.

5 ANALYSIS AND COMPARISON

This section presents the comparative analysis of the three DSL tools based on the criteria defined in section 3. Table 1 summarizes the results obtained.

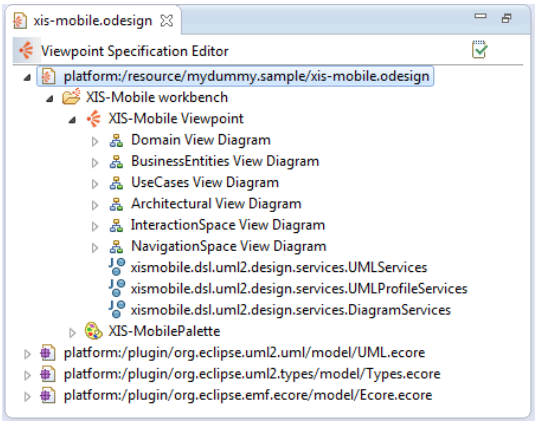

Figure 3: Viewpoint Specification Model for XIS-Mobile in Sirius.

Learnability – Papyrus proved to be the tool that required less effort to create the XIS-Mobile language and its supporting workbench. This can be explained by the fact that Papyrus only allows by itself the definition of the profile, using a regular UML Profile diagram, and the creation of custom palettes for each XIS-Mobile's diagrams by clicking on the palette toolbar. The user can then load the profile and the palettes manually, or can instead create an Eclipse plug-in project and configure the plug-in manifest defining configurations like the location of the model and its extension points. In EA the definition of the XIS-Mobile language is also achieved using a profile diagram. Unlike Papyrus, EA allows much more customization, namely it allows defining custom toolboxes, diagrams, project templates and patterns. For instance, the creation of custom toolboxes and diagrams is also performed using profile diagrams which contain special metaclasses like *ToolboxPage* or *Diagram_Logical*. In order to load all this information, the user needs to create a MDG technology file. This file can then be loaded by the user (manually) or using a plug-in (automatically). The latter option requires an extra effort on programming the load of the developed MDG technology. Lastly, Sirius is the tool that requires more effort, mainly because it defines the workbench details in a tree style view containing several layers of configurations. Unlike the previous two tools, Sirius does not provide its own editor or modeler to create and edit the DSL metamodel. This can be overcome using an Eclipse plug-in for that purpose, being even Papyrus one of the possibilities. Additionally, since Sirius provides the creation of fully customized workbenches, it also requires a

significant amount of configuration even for a small example. In the case of the XIS-Mobile, this issue becomes even more critical by the fact that it is a UML profile, requiring the querying and stereotype filtering used in particular contexts. A very positive point when compared to the other tools is that Sirius allows the test of the developed workbench on-the-fly.

Usability – In terms of usability, EA proved to be the most mature and more efficient tool, not only due to the amount of options it provides, but also to the lack of bugs it contains. Papyrus revealed itself somewhat unstable with some crashes and bugs (e.g. impossibility to delete and associate elements on occasions) caused by the fact of being a work in process project. Moreover, Papyrus proved unable to correctly apply icons or shapes to language elements, thus failing to improve the graphical semantics of the produced models. This aspect is clearly a disadvantage when compared to EA and other UML-based workbenches. Sirius has better usability than Papyrus and is more stable, but its approach of developing and configuring many aspects of the DSML in a tree style view can be cumbersome, especially the Viewpoint Specification Model that requires the use of several configurations.

Graphical Completeness – EA and Sirius were the tools that allowed to fully reproduce all the XIS-Mobile elements with fidelity. In contrast, Papyrus revealed several important limitations and bugs in the creation of the InteractionSpace view diagrams. More precisely, Papyrus did not permit to freely position widgets inside interaction spaces preventing the correct modeling of GUI layouts.

Validation Support – EA is the tool that offers by itself the most complete validation support. The offered UML profile mechanism restricts each stereotype according to its metaclass, be it XIS-Mobile stereotypes, toolboxes or diagrams. Moreover, EA allows to programmatically define model constraints that will be performed over the model. Both Papyrus and Sirius are only able to perform by themselves limited validations in the types of each DSL element by leveraging the UML profile mechanism. All the remaining verifications and the eventual triggering of error messages are only possible recurring to other Eclipse plug-ins dedicated to that purpose (e.g. EVL).

Transformation Support – Similarly to what happens in the previous criteria, only EA supports by itself M2M and M2T. Both can be specified using templates written in a proprietary DSL. The M2T transformations can only transform one element into another despite the hypothetical need to generate various elements from one. To overcome this issue, EA also allows the manipulation of models programmatically. Both Papyrus and Sirius only support model transformations when used in conjunction with other Eclipse plug-ins (e.g. Acceleo, JET or ATL).

Evolvability – Language evolution is somehow equivalent in all three tools. Papyrus, EA and Sirius need to be restarted whenever the user wishes to see the changes performed in the DSL metamodel. Papyrus can eventually update automatically old models, if the metamodel has been loaded manually. Sirius just provided a consistent behavior when it runs on-the-fly changes related with the DSL workbench. EA needs to be reloaded in either of these cases.

Interoperability – Both Papyrus and Sirius are able to import and export the produced models in EMF-compatible formats, including Ecore and XMI. EA also supports the import and export in Ecore and XMI (but with proprietary namespaces).

Summarizing, EA proved to be the most suitable tool to support the definition of a complex DSML such as XIS-Mobile language. EA offers a very good support for UML profiles not just in terms of usability, but also in terms of possibilities of customization (e.g. toolboxes, diagrams, project templates). Also the provision of integrated mechanisms to perform M2M and M2T mechanisms is quite important in every MDD framework project. Therefore, EA revealed as one of the best tools currently available for the development of a DSML based on a UML profile. However, a downside is that it is a proprietary tool. If a user intends to create a standalone DSML without leveraging any UML facility, EA may not be the most appropriate tool, since it was mainly tailored for UML-based modeling. We can conclude that EA is a clear example of a CASE tool that evolved to a language workbench by providing a set of customizations to support the creation of user-specified DSLs. Similarly to EA, Papyrus is strongly bound to UML modeling and provides good support for UML profiles. However, Papyrus revealed serious usability limitations and as so, currently is not a solution to model and maintain the XIS-Mobile language (neither DSMLs with a large amount of concepts). It can be useful to create small languages based on UML and the fact of being available for free is an important advantage. From the three tools analyzed, Sirius is the one that offers a larger degree of freedom by allowing the creation of workbenches for any EMF-based language (both textual and

graphical) representing the concepts of that language using diagrams, tables or trees. These facts make Sirius rather different from EA and Papyrus especially because it is not specifically tailored for UML neither provides a modeler to define the DSL metamodel. The use of XIS-Mobile with Sirius proved to be a painstaking task because, as pointed previously, Sirius required to be configured for taking into account the desired UML elements and stereotypes. Due to its nature, Sirius appears to be more adequate for standalone DSLs that do not use UML as its base metamodeling approach, because it is not immediately ready to support UML. Performing that configuration for languages like XIS-Mobile can be an overkill.

Table 1: Evaluation Results.

Criteria \ Tool	Papyrus	EA	Sirius
Learnability	Low	Medium	High
Usability	Medium	High	Medium
Graphical Completeness	Medium	High	High
Validation Support	Low	High	Low
Transformation Support	Low	Medium	Low
Evolvability	Medium	Medium	Medium
Interoperability	Medium	Medium	Medium

6 RELATED WORK

Previous work reports the analysis and comparison of language workbenches with different concerns and with different purposes. In general this previous work reports experiences using small and somehow simpler DSMLs or textual DSLs.

(Amyot, et al., 2006) present a study that evaluates five DSML tools (GME, Tau G2, RSA, XMF-Mosaic and Eclipse with GEF and EMF) by using the Goal-Oriented Requirement Language (GRL) as use case. Like XIS-Mobile language, GRL has a graphical representation and can be used either as a UML profile or as an independent language. However, XIS-Mobile has a wider set of concepts and relationships than GRL, being therefore more complex, which can be an important factor when choosing the tools to use. Moreover, some of the tools analyzed are proprietary and/or obsolete (e.g. Tau G2, RSA, XMF-Mosaic). Some evaluation criteria used in (Amyot, et al., 2006), such as Graphical Completeness and Editor Usability, were also used in this paper, since they were considered useful to evaluate DSL tools.

Similarly to what is done in this paper, (Saraiva & da Silva, 2008) evaluate a small set of tools that allow the creation of DSMLs and discusses some open research issues in this area. An evaluation framework is proposed with some dimensions focused either on architectural details (e.g. level compaction) and practical usage of the tools (e.g. model transformation support). The evaluation framework is applied using a small case study, the Social Network metamodel, while in this paper a more complex metamodel is used for the purpose.

Unlike this paper, (Vasudevan & Tratt, 2011) focus on textual DSL tools; for that reason, the representation of the DSLs produced with these tools is similar to a programming language. Moreover, the case study used to evaluate the tools is a finite state machine, which is considerably less complex than XIS-Mobile. The evaluation is performed using a set of "dimensions" (e.g. approach or error reporting) for qualitative evaluation and "metrics" (e.g. lines of code or aspects to learn) for quantitative evaluation. While some of these dimensions and metrics are captured by the criteria use in this paper, others like the number of lines of code only make sense for textual DSL tools evaluation.

The annual Language Workbench Challenge (LWC), launched in 2011, is another initiative that promotes the comparison and discussion of DSL workbenches. (Erdweg, et al., 2013) present and discuss ten language workbenches that participated in LWC'13. Unlike this paper, the set of tools analyzed was defined according to the tools that applied to solve an assignment and were subsequently accepted. The assignment was to implement a DSL for questionnaires, which should be rendered as an interactive GUI reactive to user input to present additional questions. Additionally, the produced DSLs did not have the restriction of being graphical, unlike in this paper. The evaluation of the tools was more exhaustive than the one presented here, since it was used a feature model that contemplates features of both textual and graphical DSL tools.

(Savić, et al., 2014) report their experience using MPS to implement a textual language, called SilabReq, for requirements specification and compare MPS with other alternative tools (Spoofax, Obeo Designer, MetaEdit+, Xtext, Papyrus and EMFText) in what respect the following criteria: support for the abstract and concrete syntax definition, and supported IDEs. However this paper analyzes more tools, it provides a shallow analysis

on those tools and is only concerned in a textual DSL.

Recently, (Morais & da Silva, 2015) proposed and used the ARENA framework to compare and evaluate user-interface modeling languages.

7 CONCLUSIONS

This paper presented an analysis and comparison of three DSL workbenches (Papyrus, EA and Sirius), which were evaluated in the context of a complex DSML, the XIS-Mobile language. Tools like the ones surveyed are becoming popular by incorporating the MDD principles and supporting the definition of DSLs, as well as the creation of their customized IDEs. These workbenches allow raising the level of abstraction not only by using concepts specific to a certain domain, but also through the use of model transformations that aim to increase the speed of the development process and time-to-market of software applications.

An evaluation framework was defined in order to better analyze these workbenches. This framework included the following criteria: learnability, usability, graphical completeness, validation support, transformation support, evolvability and interoperability. After using each one of the tools and analyzing them against these criteria, it was possible to identify their strengths and weaknesses, as well as assess their suitability to develop non-trivial languages. Additionally, this paper discussed related work allowing us to further research the complexity of these tools. This evaluation framework provides a qualitative evaluation, which can be somewhat subjective. In the future, this framework may be extended possibly including other criteria reflecting measurable aspects of language workbenches. Also its application to a wider set of workbenches can better aid identifying other important features in such tools.

ACKNOWLEDGEMENTS

This work was partially supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, under the projects POSC/EIA/57642/2004, CMUP-EPB/TIC/0053/2013, UID/CEC/50021/2013 and DataStorm Research Line of Excellency funding (EXCL/EEI-ESS/0257/2012).

REFERENCES

Amyot, D., Farah, H. & Roy, J.-F., 2006. Evaluation of development tools for Domain-Specific modelling languages. *System Analysis and Modelling: Language Profiles,* 4320(12).

Bourguignon, J.-P., 1990. Structuring for managing complexity. *Managing Complexity in Software Engineering,* 17.

Costagliola, G., Deufemia, V., Ferrucci, F. & Gravino, C., 2006. Constructing Meta-CASE Workbenches by Exploiting Visual Language Generators. *IEEE Transactions on Software Engineering,* 32(3).

Da Silva, A. R., 2015. Model-Driven Engineering: A Survey Supported by a Unified Conceptual Model. *Computer Languages, Systems & Structures,* 43.

De Sousa Saraiva, J. & da Silva, A.R., 2009. *CMS-based Web-Application Development Using Model-Driven Languages*. Proc. of ICSEA, IEEE.

De Sousa, L. M. & da Silva, A. R., 2015. A Domain Specific Language for Spatial Simulation Scenarios. *GeoInformatica*.

Erdweg, S. et al., 2013. The State of the Art in Language Workbenches. *Software Language Engineering,* 8225.

Filipe, P., Ribeiro, A. & da Silva, A. R., 2016. *XIS-CMS: a MDD Approach for Content Management Systems Modules Applications*. MODELSWARD.

Fowler, M., 2005. *Language Workbenches: The Killer-App for Domain Specific Languages?* http://www.martinfowler.com/articles/languageWorkbench.html.

Morais, F. & da Silva, A. R., 2015. *Assessing the Quality of User-Interface Modeling Languages*. Proc. of ICEIS, SCITEPRESS.

Ribeiro, A. & da Silva, A., 2014. *XIS-Mobile: A DSL for Mobile Applications.* Proc. of SAC, ACM.

Saraiva, J. D. S. & da Silva, A. R., 2008. Evaluation of MDE Tools from a Metamodeling Perspective. *Journal of Database Management,* 19(4).

Savić, D. et al., 2014. *Preliminary Experience Using JetBrains MPS to Implement a Requirements Specification Language*, Proc. of QUATIC, IEEE.

Schmidt, D., 2006. Guest Editor's Introduction: Model-Driven Engineering. *IEEE Computer,* 39(2).

van Deursen, A., Klint, P. & Visser, J., 2000. Domain-Specific Languages: An Annotated Bibliography. *SIGPLAN*, 35(6).

Vasudevan, N. & Tratt, L., 2011. Comparative Study of DSL Tools. *Electronic Notes in Theoretical Computer Science,* 264(5).