Exchanging Solutions for Information Systems Development using a Model Pattern Perspective

Diagram Templates in the Context of the Mdarte Collaborative Evolution Process

Rodrigo Salvador Monteiro[1], Geraldo Zimbrão[2] and Jano Moreira de Souza[2]
[1]*Computer Science Department, Universidade Federal Fluminense, Niterói, Brazil*
[2]*COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil*

Keywords: Collaborative Design, Sharing Solutions, Code Generation, Model Driven Architecture, MDA, MDArte, Model Patterns.

Abstract: Information Systems Development faces many recurrent issues that must be addressed in every project. A lot of common requirements and features repeatedly appear on different projects challenging the development team. Exchanging solutions and the expertise gained over the evaluation of such solutions among projects can prevent the development teams from reinventing the wheel. The MDArte framework has been used to develop information systems through the Model Driven Architecture approach, automating the generation from models reaching around 80% of the application code. Most important is that the MDArte framework turned out to be a common platform between the adherent projects used to share new solutions and features. This paper focus on recurrent issues present at the modeling phase. We applied the concept of Model Patterns in order to provide diagram templates. A set of diagram templates were designed, implemented and evaluated by a volunteer project. The feedback was extremely positive and the proposed approach proves to be very promising. This level of collaboration between different projects on one hand accelerates the development and on the other hand prevents from neglecting important issues on information systems development.

1 INTRODUCTION

When it comes to information systems development, the development teams are flooded out with business requirements, recommended practices, technological requirements, organizational patterns and so forth. Even worse, the time constraints for delivering full productive systems are getting tighter in order to cope with organizational needs. Immersed in all these worries and hurry it is a natural behavior for the development team to try to prune the requirements as much as they can. Keep it simple and stupid (kiss rule) in order to cope with the deadlines. Although it is an understandable behavior given the context, this approach leads to some problems: (1) some non-functional requirements, such as security issues, may be neglected; (2) some desirable business requirements may not be implemented due to the lack of time for designing a proper solution; and (3) some mandatory business requirements may consume too much time for defining a solution from scratch. Most of the time, when asking if someone has already been through the issue you have in hand the answer is yes. This is especially true if you are talking about how to use the underlying technology or framework that will support the development of your specific business requirement. Furthermore, if you have not defined a specific technology, library or framework yet, your research will probably reveal a suitable one. The main issue here is that, although it is a better approach than building from scratch, it still does take a considerable effort. It is not just a matter of discovering what could help. You also have to check compatibility with your environment setup, define which specific version should be used, define how it will be combined with everything else you are using and so forth. Now, what if someone has already been through all these steps in a development environment similar to yours? If he/she is willing to share the solution it would be directly applied to your case. This is exactly what the adherent projects of the MDArte framework (MDArte, 2015) have

Monteiro, R., Zimbrão, G. and Souza, J.
Exchanging Solutions for Information Systems Development using a Model Pattern Perspective - Diagram Templates in the Context of the Mdarte Collaborative Evolution Process.
DOI: 10.5220/0005690306400647
In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, pages 640-647
ISBN: 978-989-758-168-7

been doing and benefiting from the exchange of solutions through what we call the collaborative evolution process. If no one has been through your case, you will have to follow the traditional way, but if someone has gone through all the steps to reach a solution and have shared it then it will be readily available for use in your project. In this paper we applied the concept of Model Patterns in order to identify similar diagram structures. Diagram templates can be provided in order to prevent recurrent modeling tasks. This work also presents a proof of concept for applying the proposed Model Pattern approach in a MDA development environment.

This paper is structured as follows: section 2 introduces the MDArte framework and the underlying methodology; the collaborative evolution process is depicted in section 3; the concept of Model Patterns is described in section 4; section 5 and 6 discuss the use of Model Patterns respectively in the context of the Collaborative Evolution Process and in the context of the MDA transformation procedures; a proof of concept for using Model Patterns in practice is depicted in section 7; related works are addressed in section 8 and section 9 concludes the paper along with some future work.

2 MDARTE

Model Driven Architecture (MDA) is a methodology for information system development based on values from Model Driven Development (MDD, 2003). The OMG (OMG, 2015) released it in 2001, and its main goal is to formalize a methodology on code generation that uses the Unified Modeling Language (UML, 2005) and some other industry standards to build abstraction layers from model to code. These layers represent the main idea behind the MDA: promoting the separation between what is considered specific to or independent of technology. Thus, the same model could be used to generate the information system for different platforms without too much additional work. From the models, more than 80% of application code can be automatically generated; leaving around 20% to be written to fit the specific platform needs (Guttman and Parodi, 2006). Based on the fact that the generation covers almost all the code, the development requires less specific knowledge from developers and reduces both development time and cost.

After the methodology was released by the OMG, different organizations started migrating and building code generation frameworks to work with its concepts. One of the most widely known and used MDA frameworks is the AndroMDA (AndroMDA, 2015), an open source project that deals with different target technologies. In Brazil, the project was extended due to the Brazilian Government's need to standardize the development of its information systems, resulting in the creation of the MDArte framework. Today, it is an open collaborative community involving public and private sectors, from industry to academia. Besides this, although both frameworks have been exploring the generation of specific information systems, they are still contributing to each other in order to streamline code generation.

2.1 Working with the MDArte Framework

The MDArte consists of a framework with a set of cartridges that covers different application solutions. Its cartridges are components responsible for generating code for specific platforms: EJB, Hibernate, Java, jBPM, JUnit, and Struts. Because of its open community nature, the MDArte allows its users to create their own cartridges or improve existing ones. Those cartridges can be enabled or disabled depending on the need of each project to be generated, at any time. Fig. 1 illustrates how they are coupled to the framework and the code generated.

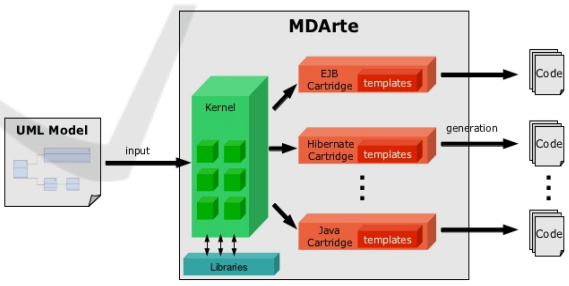

Figure 1: The MDArte structure model.

The process starts by the transformation kernel reading the UML model representing the information system specifications. After that, the kernel starts the activated cartridges triggered to each UML elements from the model (i.e., elements such as Class, Use Case, Activity and others). As a result, each cartridge called by the kernel generates a set of artifacts that can be split into two major categories: (1) application's background infrastructure and (2) implementation points. These implementation points represent places where developers will actually write code in order to cope

with the domain specific business rules or any other customization needed. The artifacts within this category are generated only once in order to prevent the loss of manual changes. The artifacts within the application's background infrastructure category are always rewritten over every model transformation. This behavior allows the automatically embedding of new features, by changing the specifications that guide the model transformation process.

3 COLLABORATIVE EVOLUTION PROCESS

The MDArte community is hosted on the Portal de Software Público Brasileiro (www.softwarepublico.gov.br) and it comprises forums, wikis, sources, tutorials, releases, etc. It serves as a central point intended to gather everybody related somehow to the MDArte. Since the beginning of its creation a clear motivation was to go beyond regular or traditional collaboration. This section presents the process that has been used in order to evolve the MDArte framework collaboratively. In a birds eye view, the MDArte forums and adherents projects are monitored by the MDArte core team in order to identify solutions designed in one project that could be of interest for others. Once a solution is identified, the project that designed it is asked to share it with the whole community. The MDArte core team analyses the designed solution in order to remove specificities from the application domain and thus generalizes it in order to improve reusability. The generalized solution is embedded into the MDArte cartridges either by changing existing model transformation procedures or by creating new ones. Finally, a new MDArte release is published in order to make the solution available for the whole community. The collaborative evolution process is depicted in Fig. 2 and explained in the following.

The first step in the process is to identify requirements or solutions of interest. Any requirement that has a reasonable chance of appearing again in other projects or a solution that could be reused represents a candidate. It is important to remark that sometimes only the requirements expressing a need are present without any solution. As the MDArte community is also composed by academic institutions, these requirements may reveal a real-world problem that deserves a research investigation. This a very important feature in the MDArte environment as it serves as a source of real-world problems that can feed and motivate academic work such as final projects, thesis and so forth. In such cases, the proof of concept prototype generated by the academic work is the input for the next step.

Either if we have in hand a whole solution provided by a project or a proof of concept prototype provided by an academic work, the next step is to generalize the solution in order to improve reusability. The main concern here is either to turn the proof of concept prototype into a complete solution solving and removing some simplifications typical for such prototypes, or turn the domain specific solution provided by one project into a generic solution. In both cases the goal is to come up with a ready-to-use and reusable solution.

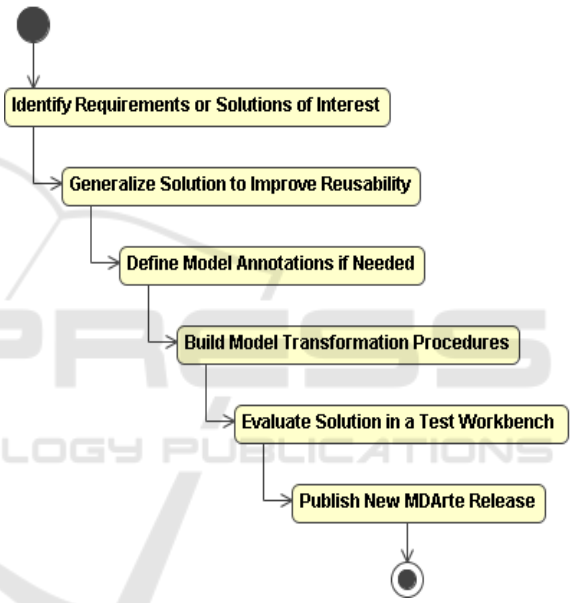

Figure 2: The Collaborative Evolution Process.

Following the process it is required to define if it will be necessary to annotate the UML model in order to trigger the generation of the solution. In many cases, the application models expressed in UML must indicate where the solution is to be applied. As an example, when requesting to publish a service component as a Web Service or to generate an audit trail for an entity manipulated by the system it is required to annotate the specific service components or entities that must present the expected behavior. However, sometimes no annotation is required as it is the case when introducing code to prevent script injection. Actually, you want to introduce this behavior for the whole set of components of your system and thus it is not required to introduce any annotation at all.

Next, the model transformation procedures must be implemented in order to embed the solution into the code being generated. If a specific annotation was defined, the new procedures are linked to them so that the procedure will be triggered only when the defined annotation is encountered.

The whole new set of model transformation procedures are tested with project examples from a test workbench in order to detect problems or incompatibility issues.

Finally, a new release of MDArte comprising the new embedded solution is published and made available for the whole community.

It is interesting to highlight that the more the adherent projects benefit from the shared features the more proactive they turn to be. Indeed sometimes a project provides not only the solution but also the model transformation procedures they have built locally to implement it, which eases the job for sharing it. Of course there are projects at complete different levels, but we try to keep a common motto: share what you have at the level you feel confident. Just please do not keep it only to yourself. It does not matter if it is just a requirement, a full solution or the model transformation procedures implementation.

4 MODEL PATTERNS

When it comes to information systems development, recurrent issues can also be identified over the models that represent the structure and behavior of the systems. For example, if we look at the UML diagrams describing a set of use cases for performing basic maintenance operations over different domain entities of a system it is expected that many of such diagrams will look very similar. Actually this similarity is promoted by some development guidelines in order to provide a standardized interface to the end user. Consider an information system in an academic environment responsible for managing information on courses, students, etc. Fig. 3 presents an activity diagram specifying the flow of a use case for querying students.

The first activity allows for capturing input filter parameters from the user. The second activity performs the filter querying the backend database and retrieves the students that conform to the filter conditions. A list of students returned by the system is presented to the user in the third activity. Finally, the user can decide among three possible actions: delete a student returned in the filter list, look into the details of a specific student or ask to perform a new filter restarting the use case. Now, if we look at the use case flow for querying courses we will find a very similar diagram structure. Beside the specific filter parameters, which would be naturally different from students to courses, it is expected that the same activities, flow and actions would be present. Indeed, in many development environments it is a concern to guarantee that such standardization is obeyed. Identifying recurrent diagram structures leads to what we call Model Patterns. In other words, Model Patterns can be defined as a set of diagrams and their inner elements representing a standardized behavior associated to higher level application functionality. Model Patterns can be viewed as templates, as they define at the same time the structure and the places where specific information should be inserted in order to derive specific instances.

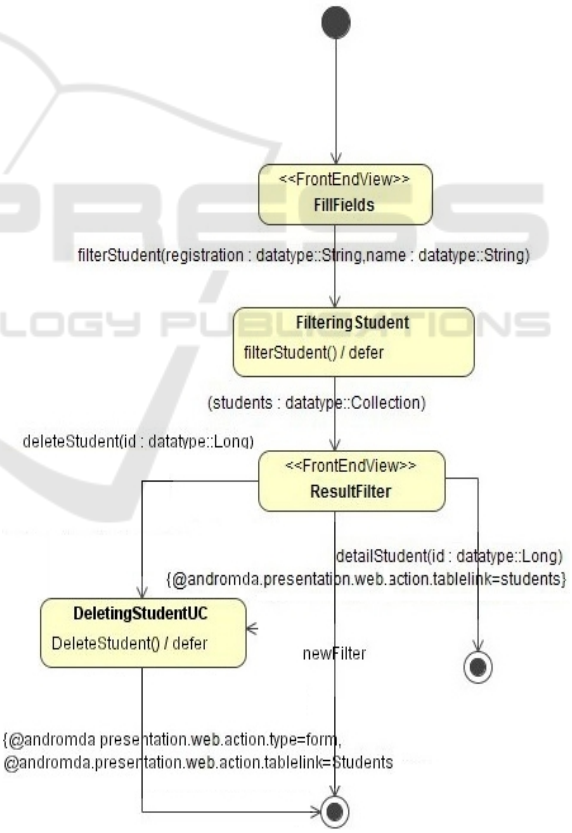

Figure 3: Activity diagram representing the query student use case flow.

5 MODEL PATTERNS IN THE COLLABORATIVE EVOLUTION PROCESS

As stated in the previous section, Model Patterns are able to capture recurrent behavior representing it through template diagrams. If we look back to the Collaborative Evolution Process depicted in Fig. 2 we can consider a Model Pattern to be a Solution of Interest. Once we agree to do so, we can ask the adherent projects to identify recurrent diagram structures over their application domains. Furthermore, in a broader view it is possible to identify cross-domain recurrent diagram structures. In the first case, activity diagrams representing CRUD (Create, Retrieve, Update and Delete) use cases are representative examples. Regarding cross-domain recurrent diagrams, we could identify diagrams for performing login and administration tasks, such as an administrator use case for resetting a user password. If we provide Model Patterns to address each one of the identified recurrent diagram structures it will massively reduce the effort of building diagrams. Furthermore, the automatically produced diagrams will also be transformed to platform-specific application code reducing the need of manual coding.

In the two following sections, we depict the general approach for embedding Model Patterns into the MDA transformation procedures and present a proof of concept implementing a specific Model Pattern for CRUD activity diagrams.

6 MODEL PATTERNS IN THE MDA TRANSFORMATION PROCEDURES

The MDArte cartridges comprise model transformations receiving models as input and providing coding artifacts as output. Although, nothing prevents us from taking models as input and producing other models as output. Actually, this conforms to the MDA approach which also considers model to model transformations. Fig. 4 sketches the regular MDArte transformation which deals with model to text transformations.

In order to embed Model Patterns into the MDA transformation procedures we must provide annotation elements, regarding the third step in the Collaboration Evolution Process, which will trigger the diagram templates associated with the desired Model Pattern. The diagram templates will be able to read the required information from the input model for filling the specific information into the templates in order to produce a specific diagram instance. The diagrams produced by the template execution should then be presented as a new input to the MDA transformation procedures in order to produce the final coding artifacts. Fig. 5 sketches the steps described above.

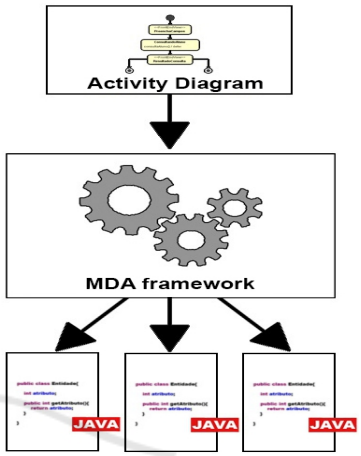

Figure 4: Sketch of Model to Text transformations.

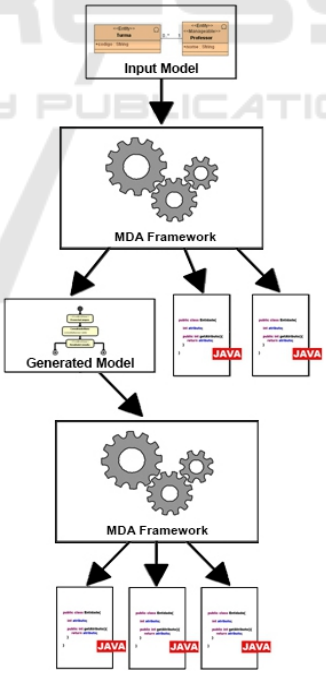

Figure 5: Sketch of Model to Model and Model to Text transformations.

7 PROOF OF CONCEPT: MODEL PATTERN FOR CRUD ACTIVITY DIAGRAMS

Most of the entities handled by an information system must have use cases for performing basic data maintenance operations, such as creating, retrieving, updating and deleting instances. This set of operations is commonly known by the acronym CRUD. Consider a project using a Model Driven approach applying activity diagrams in order to specify the use case flows. The task of designing activity diagrams for each set of CRUD use cases for each entity of the system is tedious and time consuming. Furthermore, the mechanical nature of this task invariably leads to copy-and-paste errors and when performed by different developers normally compromise the established standards.

CRUD activity diagrams are, therefore, recurrent diagram structures that could be represented by Model Patterns templates. Once the solution of interest has been identified, we must generalize the solution in order to improve reusability. In this case, it corresponds to design activity diagram templates following some desired standard use case flow, removing all information regarding a specific entity and replacing it with annotations that will be replaced latter in the process. This should be done for all CRUD diagrams. Fig. 6 presents an activity diagram template for updating instances of an entity.

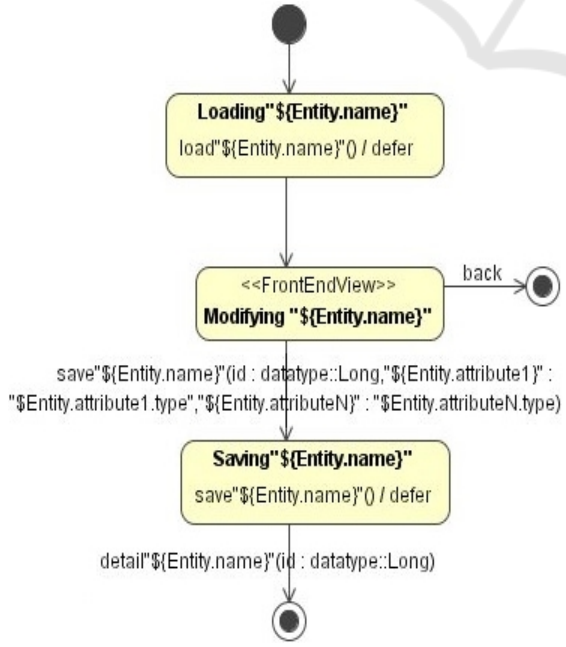

Figure 6: Activity diagram template for the Update use case flow.

Once the template is designed, we must evaluate it in order to identify the nature of the missing information. This information corresponds to the data that must be provided in order to fill the template parameters. A closer look at the template at Fig. 6 reveals that the missing information corresponds to the details of the domain entity that will be handled by the CRUD. On the other side, it should be possible to specify which domain entities must have CRUD use cases. This leads us to the need of defining model annotation in order to specify which domain entities will trigger the generation of CRUD diagrams. In this particular case, we created a new stereotype named *<<Manageable>>* which must be applied to the desired domain entities. Fig. 7 shows an example of a domain entity with the new stereotype applied.

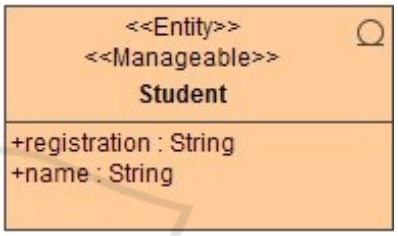

Figure 7: Example of domain entity with the manageable annotation.

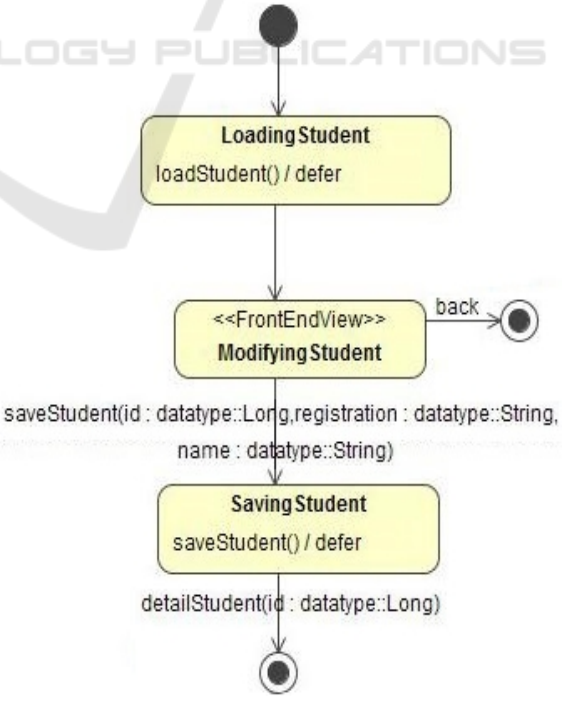

Figure 8: Update diagram instance generated for the Student class.

The MDArte model transformation specifications where evolved in order to link UML Classes with the Manageable stereotype to the new designed diagram templates. The result of applying the template in Fig. 6 to the domain entity in Fig. 7 can be seen in Fig. 8.

The same way, diagram templates for the remaining CRUD operations were designed and implemented. A new candidate release of MDArte was distributed to a volunteer project. On one hand, the project members' feedback was extremely positive, once that they could perceive in practice gains of productivity. On the other hand, it turned out to be a very promising approach, once that they could identify a lot of new candidate features, such as supporting relationships among entities. The new features identified are being analyzed and will derive new diagram templates in future releases.

8 RELATED WORK

Most of the researches regarding collaborative environments seem to be focused on the improvement of system development processes. The work presented on Ochoa et al. (2010) has explored the use of collaborative applications applied directly to the Software Engineering environment, aiming to improve its process. It introduces a collaborative application that can be used by small teams to share data related to Software Requirements. Targeting the same environment, the work on Zanoni et al. (2011) described a semi-automatic method to speed the documentation during the development process and allows the workgroup to have better visibility of the source code being produced and its documentation. Moreover, it is also possible to find significant works in the area that uses Model Driven Architecture (MDA), as described in Angelaccio and DAmbrogio (2007), to promote collaborative support between members of development environments. Moreover, when taking in consideration the works that target the final user, we have Matera et al. (2003) as one of the references related to model-driven approach for collaborative web application. Besides, in Pinel et al. (2012) we describe the use of MDA to embed collaborative tools into information systems.

Our work differs from others as it explores the advantages of the Model Driven Architecture to share solutions among different projects. This way, every project participating in the community may benefit from all solutions made available by the whole community. Moreover, no matter the specific application domain of each individual projects they are constantly challenged by recurrent issues that appear in complete different contexts. This way, even when the application domains are completely different, they can still contribute to each other.

9 CONCLUSION

The task of information system's development teams are getting more challenging. Tight schedules and rapidly changing requirements are increasingly frequent. Providing the right tools for the development team in order to face such challenges is a critical issue. The use of new methodologies such as the Model Driven Architecture approach is gradually proving its value and helping development teams to cope with their tasks. This paper presented how a virtual community built around the MDArte framework has been using MDA to exchange solutions for recurrent requirements present on information systems development. The adherent projects collaborate with the MDArte framework evolution providing from requirements to be addressed to complete model transformation procedures in order to enrich the capabilities of the framework. By doing so, the whole community is benefited. The experience over the last years has shown that the more the project is benefited by the framework the more it is willing to contribute with new solutions. The major contribution of this paper is to report this successful experience on reuse and collaboration among projects with completely different application domains. Such level of collaboration is achieved by following what we called collaboration evolution process. Through this process, the projects can exchange and share solutions by the evolution of the MDArte framework. In this paper we focused on the use of diagram templates aiming at preventing recurrent modeling tasks. The increasing level of collaboration serves as an evidence of the usability and usefulness of the MDArte framework and the presented approach. As future works we can list: report the application of the model in real world information systems developed so far; the need of proposing a business model to support and sustain the community in order to make it more independent from the institutions that conceived it; promote a tighter interaction between the adherent projects and the Academia; and aggregate new academic institutions in the community ecosystem.

ACKNOWLEDGEMENTS

Authors of this paper would like to thank FAPERJ, CAPES and CNPq for supporting part of this research.

REFERENCES

MDD, Model-driven development, *IEEE Software Special Issue*, S.J. Mellor, A.N. Clark, T. Futagami (eds.), vol 20, n. 5, September 2003.

UML, Object Management Group, Unified Modeling Language (UML): *Superstructure, version 2.0*, August 2005.

Ochoa, S. F., Quispe, A, Vergara, A., and Pino, J. A., "Improving requirements engineering processes in very small software enterprises through the use of a collaborative application", in the 2010 *14th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2010, pp. 116-121.

Zanoni, J. C., Ramos, M. P., Tacla, C. A, Sato, G. Y., and Paraiso, E. C., "A semi-automatic source code documentation method for small software development teams," in Proceedings of the 2011 15th International Conference on Computer Supported Cooperative Work in Design (CSCWD), 2011, pp. 113-119.

Angelaccio, M., and DAmbrogio, A., "A model transformation framework to boost productivity and creativity in collaborative working environments," in 2007 *International Conference on Collaborative Computing: Networking, Applications and Worksharing* (CollaborateCom 2007), 2007, pp. 464-472.

Matera, M., Maurino, A., Ceri, S., and Fraternali, P., "Model-driven design of collaborative Web applications," Software Practice and Experience, vol. 33, no. 8, pp. 701-732, 2003.

Guttman, M., and Parodi, J., Real-Life MDA: Solving Business Problems with Model Driven Architecture, 1st ed. Morgan Kaufmann, 2006, p. 224.

AndroMDA. URL: http://www.andromda.org. Last visit: Nov 2015.

OMG, Object Management Group. URL: http://www.omg.org. Last visit: Nov 2015.

MDArte. URL: http://www.softwarepublico.gov.br/dotlmlclubs/mdarte. Last visit: Nov 2015.

Pinel, R. E. A., Monteiro, R. S., Zimbrao, G., and Souza, J. M.: "Collaborative support embedded in information system through automatic code generation." In Proceedings of the 2012 *IEEE 16th International Conference on Computer Supported Cooperative Work in Design (CSCWD),* 2012, pp. 328-333.