Enabling Agile Web Development through In-Browser Code Generation and Evaluation*

Alejandro Cortiñas[1], Carlo Bernaschina[2],
Miguel R. Luaces[1], and Piero Fraternali[2]

[1] Universidade da Coruña, Databases Laboratory, A Coruña, Spain
{alejandro.cortinas, luaces}@udc.es
[2] Politecnico di Milano, DEIB, Milan, Italy
{carlo.bernaschina, piero.fraternali}@polimi.it

Abstract. Rapid evolution and flexibility are the key of modern web application development. Rapid Prototyping approaches try to facilitate evolution by reducing the time between the elicitation of a new requirement and the evaluation of a prototype by both developers and customers. Software generation, with disciplines such as Software Product Lines Engineering or Model Driven Engineering, favours the required flexibility for the development process. Nevertheless, each small change in the design of an application requires a full redeployment of complex environments in order to allow customers to test and evaluate the new configuration. In this work we present an approach that improves the development process reducing the complexity of deploying evaluation prototypes and enabling an agile development cycle. The approach can be applied using software generation and it is based on in-browser generation and evaluation. We also describe two real world tools that have integrated the proposed approach in their development cycle.

Keywords: software product lines, model driven engineering, agile software development, rapid prototyping

1 Introduction

Reducing time-to-market is a major concern in all software engineering disciplines. Software development methodologies have dealt with these problems using rapid development techniques such as agile development and continuous deployment, and automatic code generation techniques such as Model-Driven Engineering (MDE) and Software Product Line Engineering (SPLE). However, in order to allow the final user to review the software, all these techniques require that the software artefacts are deployed to a production environment. This task

* The work of the authors from UDC has been funded by MINECO (PGE & FEDER) [TIN2016-78011-C4-1-R, TIN2016-77158-C4-3-R, TIN2013-46238-C4-3-R, TIN2013-46801-C4-3-R]; CDTI and MINECO [Ref. IDI-20141259, Ref. ITC-20151305, Ref. ITC-20151247]; Xunta de Galicia (FEDER) [Ref. ED431G/01]; predoctoral research stay grant Inditex-UDC.

is often time and resource-consuming and hinders real-time interaction between the final user and the analyst in charge of eliciting requirements. To avoid this problem, rapid prototyping techniques have been proposed. However, prototypes are not the final products and it is often required a large effort to build them.

In order to further reduce time-to-market, we propose a new approach that integrates software-generation techniques with rapid development techniques to reduce drastically the deployment time of web applications. Furthermore, considering that web applications are the most popular kind of software products developed right now, and taking into account that they run on web browsers, it is possible to use the actual software product as the prototype during the development stages for the final user to review.

In order to evaluate the approach, we present two different implementations of our approach as use cases. The first use case is an academic example that uses for MDE as the software generation technique. The second use case is an industrial example of a tool developed by Enxenio[3] using SPLE.

This work is organized as follows: Sect. 2 summarizes the concepts involved in our approach; Sect. 3 explains the approach itself; Sect. 4 presents two use cases of applying it; Sect. 5 draws conclusions and future work.

2 Related work

Developing software following a classic approach is a slow and costly process. When a client orders a new product, an analyst has to understand what the client wants and, more importantly, what the client needs. From the ideas the analyst captures, the set of requirements for the product is forged and, eventually, the set of features that will comply with them. Then, the analysis and design of the new software is finished and the development team starts implementing the required features. Once the product is implemented, tested and deployed, the client can finally see the product he or she asked for. At this point, the client always requires changes, even with the best of the analysis. Therefore, a refinement stage is required, so a new analysis is made and the changes implemented. This goes on until the client is finally satisfied, an even then, the process starts again for every new feature that the product must support.

With a naked eye, the process does not seem very efficient, and several methodologies have appeared to decrease the costs in time and effort. Some of the new methodologies focus on the implementation and testing stages [3,15,21], whereas others try to improve the general workflow and the interaction between the analyst and other stakeholders. Examples of the former are SPLE and MDE, two fields based on the generation and reusing of software artefacts; an example of the latter is Rapid Development, a field which aims at avoiding overly complicated solutions.

[3] `http://www.enxenio.es`

2.1 Rapid Development

In past years many approaches have been proposed to reduce the time required to bring a product to market. They are specially useful with large and medium size projects, which require a complex and long development cycle before having a product that can be evaluated by customers and users.

Agile software development [1] is an incremental and iterative approach which aims at increasing productivity and adherence to requirements, while keeping the process as lightweight as possible. As an example, workflows such as SCRUM [21] simplify the introduction of new functionalities by organizing the work in small tasks and iterative sprints which reduce the time required to develop, integrate and test them.

Continuous deployment [10] is an approach which aims at immediately deploying software to customers as soon as new code is developed. It has great advantages like: new business opportunities, reduced risk for each release, and preventing the development of wasted software. It has been successfully implemented by many companies like Facebook, Microsoft, and IBM.

2.2 Software Product Line Engineering

Traditionally, the development of every software product goes through a series of steps: elicitation of requirements, design, implementation, testing and maintenance. Following this approach, if a software development company has to build a family of products, all the stages mentioned must be done for each one of them, even when the products share functionalities or are focused in the same specific market. The downside of this approach is that it requires high development and maintenance costs in order to produce high quality products, while the *time-to-market* for each product is long since development starts from scratch.

In other classic manufacturing industries, such as the automotive or the textile, the way the products are built evolved from a manual manufacturing process to industrial processes that use proper machinery. Software Product Lines Engineering is a discipline that aims at applying the same kind of evolution to the way software is built, i.e., applying mass-production, mass-customization and reuse strategies to software development. A Software Product Line (SPL) is a family of software products sharing features developed from a common set of reusable core assets that can be combined and configured in different ways for different products [11]. A SPL separates the development of these core, reusable assets (i.e., the *platform*), and the development of the actual applications (i.e., the *products*). The family of products is modelled as a set of *features*, which are end-user visible aspects or characteristics of a software system [12]. Features can be mandatory or optional, and each product is built from a selection of features made by an *analyst*. The main advantages of SPL are the reduced costs and the improved quality of the products, and the drastic reduction of the time-to-market for new products compared to the traditional approaches [20]. All these advantages are simply derived from the fact that the software assets are shared between all the products, so they are implemented once but used and tested in every product.

There are several types of SPL regarding the way the products are generated [4]. In some cases, the product is not expected to be modified after the generation, or even it is automatically deployed. In other cases, the product is generated as independent source code that can be extended or refined by a development team, and that has to be manually deployed afterwards. In the latter cases, the cost in time and effort from the generation of the source code to the deployment may be high.

2.3 Model Driven Development

Model Driven Development (MDD) is the branch of software engineering that advocates the use of *models*, i.e., abstract representations of a system, and of *model transformations* as key ingredients of software development [15]. With MDD, developers use a general purpose (e.g. UML [2]) or domain specific (e.g., IFML [18]) modelling language to portrait the essential aspects of a system, under one or more perspectives, and use (or build) suitable chains of *transformations* to progressively refine the models into executable code.

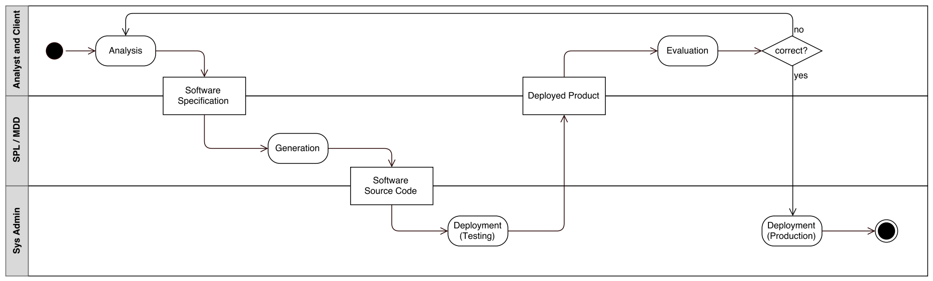
Agile Model Driven Development has been advocated as a promising approach [3], which has not yet fully expressed its potential [17]. Its idea is to organize the MDD process in ways that take advantage of the agile development principles:

1. Enabling an incremental and iterative development cycle by using tool chains able to test and validate even incomplete models [13].
2. Applying a Test-Driven Development, a distinctive feature of extreme programming [5], to MDD. [22].
3. Merging agile workflows such as SCRUM [21] with MDD in novel methodologies, to achieve system-level agile processes [23].

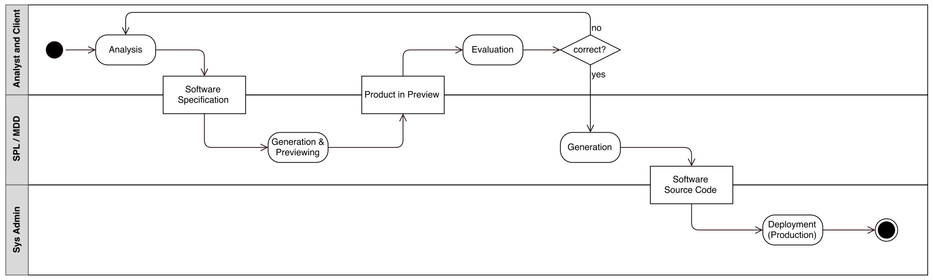
3 Our approach

SPLE, MDD or similar methodologies for the semi-automatic generation of software tackle intrinsic repetitive structures in the development of software products, either full software components like the ones assembled within a SPL, or actual generation of code from high level descriptions and model-to-model/code transformations like in MDD. These approaches are particularly suited to the iterative evolution of the project, by means of small improvements which can be easily tested and validated. However, the nature of web applications makes the evaluation of these new features complex, due to the fact that a new full deployment is necessary after each change.

Continuous deployment is particularly suited to bring these new functionalities to a set of evaluators, but it is not suitable for an integration in the development cycle itself due to its complexity. The main problematic in trying to integrate an evaluation of the final product within SPLE or MDE is related to the full code generation intrinsic in these approaches. Changes to the final product cannot be applied directly to the deployed application, but instead the high level description needs to be updated and a full code generation triggered.



(a) Original



(b) After applying our approach

Fig. 1: Activity diagram of the development process

We show in Fig. 1a a simplified workflow with processes and stakeholders involved in the development of a web application with automatic or semi-automatic generation of software. We can see how the analyst, with input from the client, elaborates the *Software Specification*; then the generation of the product, which can be based on SPLE, MDD or any other methodology, begins. The generated software has to be deployed for evaluation by a different actor, the system administrator. At this point, the analyst and the client can, together, evaluate the product and determine whether the product is the one required by the client or some changes are required. If the product is finished, then the system administrator can deploy the final version into production environment and the process ends. If the product is not complete, then the process starts again and the system administrator is involved, again, to make a redeployment in the testing environment. There are some cases when the analyst is the one that deploys the product for evaluation. In these cases, even when there is not an extra actor involved, this deployment still needs a complex environment and it is time consuming.

We propose an approach to simplify and improve the workflow for the development of web applications through automatic generation techniques by removing the necessity of an actual deployment during the development process. Only when the analyst decides that a project is correct, the full-source is generated and deployed.

The workflow, when our approach is applied, is shown in Fig. 1b. It is analogous to the previous one but, in this case, the same tool that generates the product provides a preview of it to the analyst and/or to the client. Therefore, it does not involve a real deployment of the product at this point, so the third actor, a system administrator, is not required, nor a complex environment or any extra tool for the evaluation deployment of the project. This tool we are talking about is nothing but a web browser.

Modern web browsers are able to execute code generation frameworks by means of Javascript code, and they are able to fully or partially execute the generated applications by means of *iFrame*s and *WebWorker*s. Of course, we cannot expect that absolutely every feature of the product can be previewed this way but, using mocks and similar techniques, the client can have a real idea of how the product is and introduce the required changes on the earliest stage. The generated application can be evaluated directly inside the browser following three different strategies, complemented using mocks responding to any XHR request.

Code execution. Applications fully developed with Javascript can be fully executed inside the browser. The client-side part of the application is executed inside an iFrame which is able to fully resemble a standalone browser. The server-side part of the application is executed inside a properly instrumented WebWorker able to resemble a NodeJS environment.

Full emulation. Applications developed with different technologies can exploit the full code generation intrinsic to SPLE and MDD to trigger a different version of the transformation. This way, it can generate a functionally equivalent version of the application in Javascript which can be tested using the previous strategy.

Partial emulation. Applications developed with a mixture of Javascript and other technologies can use a mixed strategy; this is, the non Javascript parts can be replaced by a functionally equivalent version and then the product is executed within an iFrame.

All the three strategies can be used to evaluate the final application without the need of complex server side deployed infrastructures, increasing productivity and reducing tools configuration complexity.

We have tried our approach within two different tools, one based in SPLE and the other one in MDD, but our approach can be applied to any other software generation technique as long as the engine is built with Javascript, which is the programming language that can be run on a web-browser[4]

[4] In any other case our approach is still conceptually valid but the generation must occur in the server side and after it, the source code must be loaded within the web client.

4 Implementations

In this section we present two academic (see Sect. 4.1) and industrial (see Sect. 4.2) tools which apply the proposed approach to real world scenarios.

4.1 IFMLEdit

IFMLEdit.org[5] is an online environment for the specification of IFML models, the investigation of their properties by means of a mapping to Place Chart Nets [14], and the generation of code for web and mobile architecture.

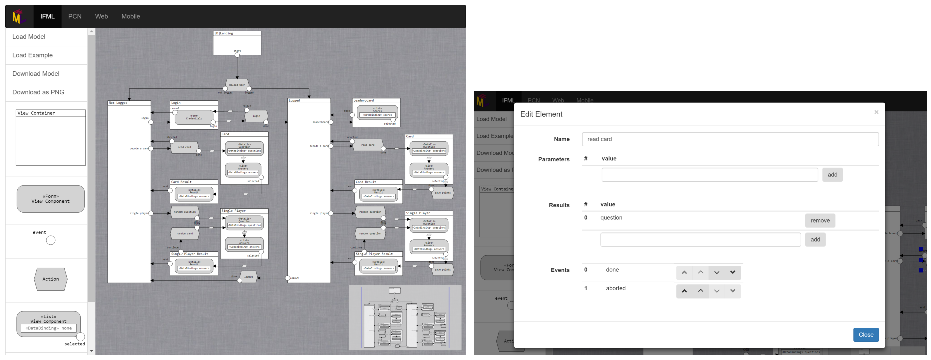
IFML (Interaction Flow Modeling Language [18]) is an OMG standard that supports the platform-independent description of graphical user interfaces (UIs) for devices such as desktop computers, laptops, mobile phones, and tablets. IFML focuses on the structure and behaviour of the application as perceived by the end user, and references the data and business logic aspects insofar they influence the user's experience, i.e., the domain objects that provide content displayed in the interface and the actions triggered from the interface.

IFML allows developers to specify the following aspects of an interactive application:

- **The view structure and content:** the general organization of the interface is expressed in terms of *ViewElements*, along with their containment relationships, visibility, and activation. Two classes of ViewElements exist: *ViewContainers*, i.e., elements for representing the nested structure of the interface, and *ViewComponents*, i.e., elements for content display and data entry. *ViewComponents* that display content have a *ContentBinding*, which expresses the link to the data source.
- **The events:** the occurrences that affect the state of the user interface, which can be produced by the user's interaction, the application, or an external system.
- **The event transitions:** the consequences of an event on the user interface, which can be the change of the *ViewContainer*, the update of the content on display, the triggering of an action, or a mix of these effects. *Actions* are represented as black boxes.
- **The parameter binding:** the input-output dependencies between *ViewElements* and *Actions*.

The tool [9], which is developed using ALMOsT.js [6], supports the following workflow: 1) the developer edits the IFML model of the application in the online editor, possibly providing hints for the generation of the fast prototype (e.g., sample data); 2) he (optionally) maps the model into a PCN and simulates the network to understand the dynamics of the application in response to events; 3) he generates the code of a fast prototype, for the web or for a cross-platform mobile language, executes and validates the prototype; 4) he turns the validated prototype into a real app, by customizing look&feel and replacing mock-up data access and operational APIs calls with real ones.

[5] `http://www.ifmledit.org`



Edit Element

Name	read card
Parameters	# value
Results	# value
	0 question
Events	0 done
	1 aborted

(a) Model editor (b) Element property editors

Fig. 2: IFML editing

IFML model editing. Figure 2a shows how the integrated IFML editor allows the developer to compose and edit the model by means of drag&drop from the palette on the left side.

Data-bindings. Once the structure of the application is modelled, the developer can use the property editor (Figure 2b) to specify how *ViewComponents* connect between them and to the data sources.

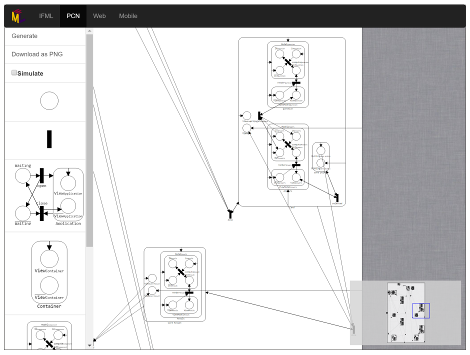
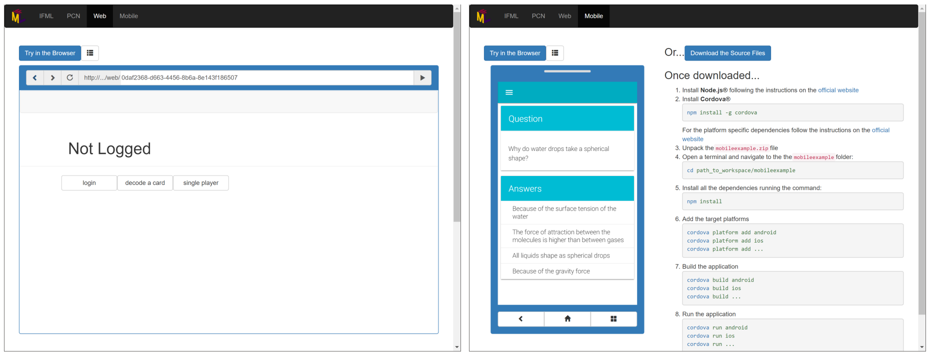


Fig. 3: Model semantics simulation

Model semantics and simulation. The developer can generate a formal description of the application by running the model-to-model transformation from IFML to PCNs. The application behaviour is rendered visually by means of tokens moving in the net, displaying the control flow in the interface and the change of status of *ViewElements*. Fig. 3 illustrates the PCN model generated from the IFML diagram of Fig. 2a; the PCN simulation helps the developer identify inconsistencies in the specified application, such as unreachable states and race conditions.


IFML
PCN
Web
Mobile
Try in the Browser
http://.../web/ 0daf2368-d663-4456-8b6a-8e143f186507
Not Logged
login
decode a card
single player
IFML
PCN
Web
Mobile
Try in the Browser
Question
Why do water drops take a spherical shape?
Answers
Because of the surface tension of the water
The force of attraction between the molecules is higher than between gases
All liquids shape as spherical drops
Because of the gravity force
Or...
Download the Source Files

Once downloaded...

1. Install **Node.js®** following the instructions on the official website
2. Install **Cordova®**
```
npm install -g cordova
```
For the platform specific dependencies follow the instructions on the official website
3. Unpack the `mobileexample.zip` file
4. Open a terminal and navigate to the the `mobileexample` folder:
```
cd path_to_workspace/mobileexample
```
5. Install all the dependencies running the command:
```
npm install
```
6. Add the target platforms
```
cordova platform add android
cordova platform add ios
cordova platform add ...
```
7. Build the application
```
cordova build android
cordova build ios
cordova build ...
```
8. Run the application
```
cordova run android
cordova run ios
cordova run ...
```

(a) Web code generation (b) Mobile code generation

Fig. 4: Code generation

Code generation. The developer can generate a fully functional prototype for both the web and mobile architecture, launching a model-to-code transformation. Figure 4a shows the generated web prototype, run on top of the web server emulated inside the browser. Figure 4b shows the generated mobile prototype, run within the mobile emulator inside the browser. In-browser emulation allows the developer to test the current web or mobile release of the prototype without installing any web server and also in absence of the Internet connection. The *Browser-Server emulator* is a pure Javascript component able to emulate a web browser, a Node.js server and the whole request response cycle that connects the two. It is used to support online and offline work seamlessly. The *Mobile emulator* is a Javascript component able to emulate a mobile cross-platform environment (now Cordova); it supports the instantaneous execution of the generated cross-platform mobile code.

Prototype download. The previous steps can be reiterated to evaluate different application structures (e.g., single vs multiple pages) and interaction approaches (e.g., update on object selection vs explicit update events). The generated prototype can be downloaded and refined to produce the final application. Each IFML Action and ViewComponent data query is encoded as a web service, which can be replaced by an external implementation.

4.2 GISBuilder

Enxenio[6] is a Spanish SME (small and medium enterprise) with expertise in GIS. Enxenio has collaborated with the Database Laboratory at the University of A

[6] `http://www.enxenio.es`

Coruña many times in the past, and several works, such as [7,16,19], are the result of this collaboration. For some time now, Enxenio and the Database Laboratory have been working on the design of a SPL for the automatic generation of web-based GIS applications [8].

GISBuilder is an internal tool that provides a web interface where an analyst can design and generate the source-code of web-based GIS. The GISBuilder initial architecture is shown in Fig. 5a and described in [8]. A brief summary of its workflow follows.

When a client comes in for a new application, the analyst determines which are the requirements of the new application and designs the application itself. Then he or she interacts with the **specification interface** and configures the product according to this design. In the specification interface three aspects of the application are set:

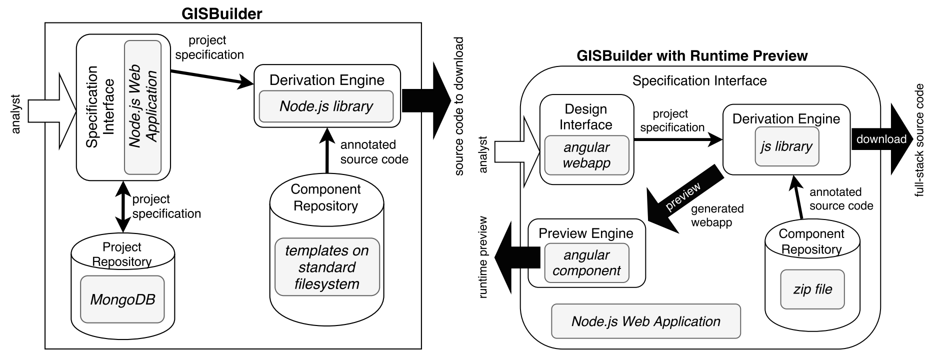
– Which features the application provides. Examples of features are *csv importer* or *user management*.
– The data model for the application: entities, properties and relationships. From this data model, the analyst can define lists, forms or map viewers, and link these elements with menu entries.
– Several aspects of the graphical user interface, such as the menu configuration, the static pages or the UI layout.

Once the configuration or specification of the product is done, the **derivation engine** is invoked. This engine takes the product specification and assembles/-generates the source code of the final products, taking the required components from the **component repository**. Since the derivation engine is based on scaffolding, the different components are nothing but annotated source code files, or templates. Furthermore, the product specification is also stored in the **project repository** so it can be reloaded and edited in the future if required.

The output provided by GISBUilder is a ZIP file with the source code of the product that has to be manually deployed within a web server with some previously installed software (such as Java, Tomcat, Node.js, npm, PostgreSQL with PostGIS, etc.). Even if it has been deployed previously, it has to be fully redeployed again. This process is slow and it usually requires more than one person, since the analyst is not in charge of preproduction deployments. In the case a client is providing feedback to the analyst, this redeployment causes the interaction with him to be slow and absolutely not in real-time.

We wanted to facilitate the interaction between the client and the analyst of the company by allowing the client to propose and evaluate changes to the product in real-time while it is being configured. To achieve this, we have applied the approach presented in this paper and we have enhanced GISBuilder to generate and show a preview of the designed products at runtime, directly on the browser, without the need of any server-side structure.

In order to get this, we have changed the way GISBuilder is designed, as we can see in Fig. 5b. The main changes are:



(a) Original architecture (b) Adapted architecture

Fig. 5: Architecture changes in GISBuilder

1) We have implemented a new version of the **derivation engine** that is able to run entirely on the web browser.
2) The **derivation engine** is integrated within the **specification interface**, as well as the **component repository**, provided as a ZIP file.
3) Our preview component intercepts XHR request of the previewing application and returns mocks responses to each specific REST petition.
4) GISBuilder produces full-stack web applications with Spring in the server side and Angular in the client side. In the adapted version, GISBuilder creates two different versions of the products, depending on whether the analyst wants to preview them or to download the full-stack version.

This way, after a reconfiguration of the product, the analyst can simply run its preview and show the client its aspect. Of course, the previewing component does not run every feature of the SPL, but it can still show enough to provide the customer a realistic view of the application. When the client is satisfied with the preview, the actual full-stack version of the product can be generated and deployed, just as before.

5 Conclusions and Future Work

Reducing the time required to introduce a new functionality is a key requirement of modern software development. Approaches like Software Product Lines Engineering and Model Driven Engineering exploit recurrent structures through high-level descriptions that are refined into final product via full code generation.

In web-based development tools, the ability to generate the code and to execute or emulate the final application in the browser enables iterative development cycles based on the evaluation of the final product after small changes without the need of complex infrastructures or development environments. We have proposed in this paper an approach that can be used to modify software development methodologies to enable agile web development through in-browser code generation and evaluation. We have also used the approach in two real world tools: an academic tool that was developed from scratch, and an industrial tool whose functionalities were extended.

In future works we will propose a standard framework aiming at facilitating the integration of the proposed approach into existing tools.

References

1. Principles behind the Agile Manifesto. http://agilemanifesto.org/principles.html
2. UML unified modeling language. `www.uml.org/`, accessed: 2017-1-10
3. Ambler, S.W.: Agile model driven development is good enough. IEEE Softw. 20(5), 71–73 (Sep 2003), `http://dx.doi.org/10.1109/MS.2003.1231156`
4. Apel, S., Batory, D., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines (2013), `http://www.springer.com/us/book/9783642375200`
5. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2000)
6. Bernaschina, C.: (ACCEPTED) ALMOsT.js: an agile model to model and model to text transformation framework. In: Proceedings of the 2017 International Conference on Web Engineering. ICWE'17 (2017)
7. Brisaboa, N.R., Cotelo-Lema, J.A., Fariña, A., Luaces, M.R., Parama, J.R., Viqueira, J.R.R.: Collecting and publishing large multiscale geographic datasets. Software: Practice and Experience 37(12), 1319–1348 (oct 2007), `http://onlinelibrary.wiley.com/doi/10.1002/spe.807/abstract`
8. Brisaboa, N.R., Cortiñas, A., Luaces, M.R., Pedreira, O.: GISBuilder: a framework for the semi-automatic generation of web-based geographic information systems. Proceedings of the 20th Pacific Asia Conference on Information Systems (PACIS 2016) (2016)
9. Carlo, B., Sara, C., Piero, F.: (accepted)IFMLEdit.org: a web tool for model based rapid prototyping of web and mobile applications. In: Proceedings of the International Conference on Mobile Software Engineering and Systems. MOBILESoft '17 (2017)
10. Claps, G.G., Svensson, R.B., Aurum, A.: On the journey to continuous deployment: Technical and social challenges along the way. Information and Software Technology 57, 21 – 31 (2015), `http://www.sciencedirect.com/science/article/pii/S0950584914001694`
11. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley (2002), `https://books.google.es/books/about/Software{_}Product{_}Lines.html?id=tHGFQgAACAAJ{&}pgis=1`
12. Kang, K.C., Cohen, S.G., Hess, J.a., Novak, W.E., Peterson, a.S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Distribution 17(November), 161 (1990), `http://www.sei.cmu.edu/reports/90tr021.pdf`
13. Kirby, Jr., J.: Model-driven agile development of reactive multi-agent systems. In: Proceedings of the 30th Annual International Computer Software and Applications Conference - Volume 02. pp. 297–302. COMPSAC '06, IEEE Computer Society, Washington, DC, USA (2006), `http://dx.doi.org/10.1109/COMPSAC.2006.144`
14. Kishinevsky, M., Cortadella, J., Kondratyev, A., Lavagno, L., Taubin, A., Yakovlev, A.: Coupling asynchrony and interrupts: Place chart nets. In: Application and Theory of Petri Nets 1997, 18th International Conference, ICATPN '97, Toulouse, France, June 23-27, 1997, Proceedings. pp. 328–347 (1997)

15. Kleppe, A., Warmer, J., Bast, W.: MDA explained - the Model Driven Architecture: practice and promise. Addison Wesley object technology series, Addison-Wesley (2003), `http://www.informit.com/store/mda-explained-the-model-driven-architecture-practice-9780321194428`
16. Luaces, M.R., Pérez, D.T., Fonte, J.I.L., Cerdeira-Pena, A.: An Urban Planning Web Viewer Based on AJAX. In: Vossen, G., Long, D.D.E., Yu, J.X. (eds.) Web Information Systems Engineering - WISE 2009. pp. 443–453. Lecture {Notes} in {Computer} {Science}, Springer Berlin Heidelberg (oct 2009), `http://link.springer.com/chapter/10.1007/978-3-642-04409-0{_}43`
17. Matinnejad, R.: Agile model driven development: An intelligent compromise. In: Proceedings of the 2011 Ninth International Conference on Software Engineering Research, Management and Applications. pp. 197–202. SERA '11, IEEE Computer Society, Washington, DC, USA (2011), `http://dx.doi.org/10.1109/SERA.2011.17`
18. OMG: Interaction flow modeling language (ifml), version 1.0. `http://www.omg.org/spec/IFML/1.0/` (2015)
19. Places, Á.S., Brisaboa, N.R., Fariña, A., Luaces, M.R., Paramá, J.R., Penabad, M.R.: The Galician virtual library. Online Information Review 31(3), 333–352 (jun 2007), `http://www.emeraldinsight.com/doi/full/10.1108/14684520710764104`
20. Pohl, K., Böckle, G., Linden, F.V.D.: Software Product Line Engineering: foundations, principles and techniques, vol. 49 (2005), `http://www.springerlink.com/index/10.1007/3-540-28901-1`
21. Schwaber, K., Beedle, M.: Agile software development with Scrum. Prentice Hall (2002)
22. Stahl, T., Voelter, M., Czarnecki, K.: Model-driven software development: technology, engineering, management (2006)
23. Zhang, Y., Patel, S.: Agile model-driven development in practice. IEEE Softw. 28(2), 84–91 (Mar 2011), `http://dx.doi.org/10.1109/MS.2010.85`