Online Model Editing, Simulation and Code Generation for Web and Mobile Applications

Conference Paper · May 2017

DOI: 10.1109/MiSE.2017.1

CITATIONS
9

READS
183

3 authors:

Carlo Bernaschina
Politecnico di Milano
22 PUBLICATIONS **63** CITATIONS
SEE PROFILE

Sara Comai
Politecnico di Milano
148 PUBLICATIONS **1,962** CITATIONS
SEE PROFILE

Piero Fraternali
Politecnico di Milano
278 PUBLICATIONS **5,977** CITATIONS
SEE PROFILE

Online Model Editing, Simulation and Code Generation for Web and Mobile Applications

Carlo Bernaschina
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano, Italy
carlo.bernaschina@polimi.it

Sara Comai
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano, Italy
sara.comai@polimi.it

Piero Fraternali
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano, Italy
piero.fraternali@polimi.it

***Abstract*—The wide range of different platforms for web and mobile applications development requires fast prototyping and the evaluation of multiple releases. Model Driven Development (MDD) represents the application at high level with textual or visual languages and can be used to automatically generate the final product by means of model-to-model and model-to-code transformations. We describe IFMLEdit.org, an open-source, online MDD tool for the specification and automatic generation of fast prototypes of Web and mobile application from IFML (Interaction Flow Modeling Language) specifications. The tool also supports the semantic mapping from IFML to PCN, a variant of Petri Nets, for model simulation and checking.**

***Keywords*-Model-driven development, Computer aided software engineering, Mobile applications**

I. Introduction

In web and mobile development, the wide range of coding platforms and device screens requires the ability to rapidly evolve and evaluate multiple releases. In the mobile field, cross-platform tools allow the creation and distribution of apps for multiple targets; with environments such as Appcelerator Titanium [1], IBM MobileFirst Platform Foundation [2], PhoneGap [3], and AppBuilder [4] developers write code once and derive the implementation for different targets. More recently, Rapid Mobile Application Development (RMAD) tools, such as Appery.io [5], Codiqa [6], Eachscape [7], Alpha Anywhere [8], Anypresence [9], show a trend towards no-code or low-code environments: they support rapid mobile development with online visual IDEs; however, for complex or customized behaviors, programming skills are still needed to code the logic behind the visual components.

An alternative approach exploits the Model Driven Development (MDD) paradigm [10], [11]. The application is represented at high level by means of a textual or visual language and model-to-model and model-to-code transformations generate the final product. MDD differs from cross-platform development and visual IDEs in that it shifts the development effort from code to models and transformations, promoting the deployment of early prototypes and mitigating the divergence between specifications and implementation typical of fast evolving application requirements. Recent MDD approaches specific for cross-platform mobile applications are surveyed in [12]: they are mainly based on textual DSLs, rely on ad-hoc, informally specified modeling languages and transformations, and are currently in a prototypical status. In this paper, we describe IFMLEdit.org, an open-source, online MDD tool for the specification and automatic generation of fast prototypes of Web and mobile applications from IFML diagrams. IFML (Interaction Flow Modeling Language) [13] is an OMG standard that supports the platform-independent description of graphical user interfaces for devices such as desktop computers, laptops, mobile phones, and tablets. IFMLEdit.org allows developers to edit or import IFML models, supports the semantic mapping transformation from IFML to PCN [14], a variant of Petri Nets, for model simulation and checking; finally, it performs model-to-text generation of fully functional web and mobile application prototypes, which can be quickly evolved into final products.

The paper is organized as follows: Section II briefly surveys the related work; Section III provides an overview of IFML; Section IV defines the formal semantics of IFML by mapping it to PCN, while Section V describes the online environment for the specification and automatic generation of Web and mobile applications; finally, Section VI draws the conclusions.

II. Related Work

MDD and Domain Specification Languages (DSL) are the key ingredients of several solutions that support cross-platform development: with MDD, DSLs are automatically transformed into working software applications by generating code or directly interpreting / executing the model.

Early approaches developed their DSLs and code-generators based on Xtext framework and Eclipse [15], like e.g., Applause [16] and Canappi (currently BuildUp App Builder) [17]. However, Xtext is typically used for simple use cases and projects [18]. More recents proposals based on textual DSLs include mD^2 [19], which produces native Android and iOS apps, and AXIOM [20], where an Abstract Model Tree is the basis for all model transformations and code generation.

Some online model-driven environments are available too: the system in [21] supports GUI design of mobile applications; however, only simple behaviors can be specified. RAPPT [22] is a bootstrapping tool that generates the scaffolding of a mobile app based on a high level description specified in a textual DSL, and requires manual coding of application logic.

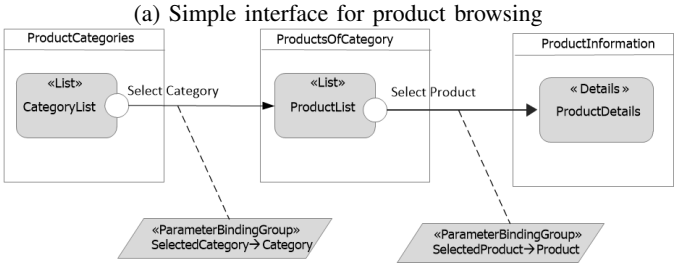
Among the commercial tools, MENDIX [23] generates hybrid applications from BPMN microflows, pre-built application components, and UI layouts; it relies on model interpretation. Outsystems [24] generates standard Java or .NET applications from the entity-relationship model for data, a FlowChart-like language for business logic and an UI based on a WYSIWYG approach. WebRatio [25] generates code from IFML [13] and BPMN [26], supporting native components, like camera, geolocation, etc.; it relies on Phonegap [3] for the cross-platform deployment. Differently from WebRatio we map IFML to Place Chart Nets (PCN) and derive an executable application prototype from the PCN, which allows both to simulate the behavior of the application on the PCN itself and to execute the original model on a simulated user interface containing all the elements and the interactions of the final application.

III. Interaction Flow Modeling Language

IFML [13] is an OMG standard that focuses on modeling the structure and behavior of the application as perceived by the end user, referencing the data and business logic aspects insofar they influence the users experience, i.e., the domain objects that provide content displayed in the interface and the actions triggered from the interface.

Figure 1 shows the IFML model for a simple product browsing interface. The interface in Figure 1a comprises

(a) Simple interface for product browsing



(b) IFML model

Fig. 1: IFML model of a simple interface

three screens: one displaying the product categories of an hypothetical online store, one listing the products of a specific category, and one publishing the details of a selected product. The IFML model, shown in Figure 1b, represents the view structure by means of three *ViewContainers* (ProductCategories, ProductOfCategory, and ProductInformation), reflecting the top-level organization of the GUI in the three screens. ViewContainers include *ViewComponents*, which denote the actual content of the interface. The generic ViewComponent classifier can be stereotyped, to express different specializations, such as lists, object details, data entry forms, and more. In Figure 1b, the ProductCategories ViewContainer comprises one ViewComponent called CategoryList, which represents the screen content, in this case a list of product categories. Also ProductOfCategory shows a list of elements, the products of the selected category, while ProductInformation displays the details of a single product.

ViewContainers and ViewComponents can be associated with *Events*, to express that they support the user interaction. The effect of an Event is represented by a *NavigationFlow*, which connects the Event to the ViewContainer or ViewComponent affected by it. A NavigationFlow is denoted as an arrow, connecting the Event associated with the source ViewElement to the target ViewElement. The NavigationFlow specifies a change of state of the user interface: the target ViewElement of the NavigationFlow is brought into view, after computing its content; the source ViewElement of the NavigationFlow may remain in view or switch out of view depending on the structure of the interface. In Figure 1b, the NavigationFlow associated with the SelectCategory Event, represented as a circle, connects the ViewComponent CategoryList with the target ViewComponent ProductList. When the Event occurs, the content of the target ViewComponent is computed so to display the product list of the chosen category; the triggering of the SelectCategory Event causes the display of the ProductOfCategory ViewContainer, with the enclosed ViewComponent, and the replacement of the ProductCategory ViewContainer, which gets out of view.

ViewComponents can have *input and output parameters*. For example, a ViewComponent that shows the details of an object has an input parameter corresponding to the identifier of the object to display; a list of items exports as output parameter the identifier of the selected item. In Figure 1b, the input-output dependency between CategoryList and ProductList is represented as a parameter binding (denoted by the IFML ParameterBindingGroup element): the value of the SelectedCategory parameter, which denotes the object selected by the user, is associated with the value of the input parameter Category, requested for the ProductList ViewComponent.

ViewContainers may embed multiple ViewComponents and also nested ViewContainers, to specify more complex behaviors and different organizations of the interface. For example, Figure 2 contrasts different GUI organizations of two different applications: a rich-client mail application (a) consists of a top-level ViewContainer with embedded sub-containers at different levels; an e-commerce web site (b) organizes the user interface into different independent ViewContainers corresponding to Web page templates.

More examples are described in the next section. For further details about IFML, the reader may refer to [13].

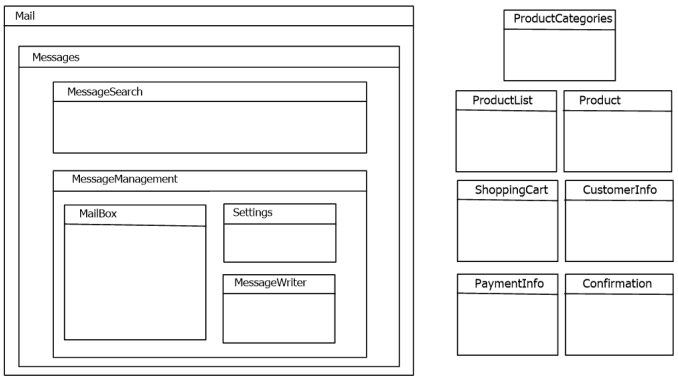


Fig. 2: Different ViewContainers for interface organization

IV. Semantic Mapping of IFML

The IFML OMG standard defines the semantics of the language informally. However, MDD tools development and interoperability requires MDD languages endowed with a precise semantics, preferably expressed formally. In this paper, the semantics of IFML is given by mapping IFML diagrams to Place Chart Nets (PCN) [14], a variant of Petri Nets characterized by compact modularization primitives.

The base construct of a PCN is a *place chart (PC)*, which represents a hierarchy of places. A PC with no parent nor children is called a *place*, because it is equivalent to a PN place. A place with a parent, but no children is called a *bottom place chart*. A place with children, but no parent is called a *top place chart*. The number of tokens in a place chart with no children is defined as in Petri Nets, while the number of tokens in a place chart with children is defined as the maximum of the number of tokens in its children. Transitions remove tokens from a group of place charts and add tokens to others. A transition is enabled when all the source place charts have at least the number of tokens required. Removing a token from a place chart with no children decrements the number of tokens by one, while removing a token from a place chart with children empties all of them. To avoid non-determinism, it is possible to insert tokens only in place charts with no children. As a way to reduce the combinatorial explosion of arcs, *default arcs* are introduced, which connect a parent place chart to one or more of its descendants. If default arcs are defined from a place chart X to (a subset of) its descendants $Y_1 \dots Y_n$, then an arc targeting X is equivalent to a set of arcs targeting the descendants $Y_1 \dots Y_n$.

The simplest IFML model is the empty diagram: it denotes an application that, as soon as opened, terminates without displaying any interface nor performing any action. Figure 3 shows the PCN corresponding to an empty IFML model.

It contains two places (Waiting and $\overline{\text{Waiting}}$), a top place chart (Application) with two children ($\text{View}_{\text{Application}}$ and $\overline{\text{View}}_{\text{Application}}$). The $\text{View}_{\text{Application}}$ bottom place chart is initialized by default from the parent. The PCN also contains two transitions called open and close. The initial marking comprises one token in Waiting and one token in $\overline{\text{View}}_{\text{Application}}$,

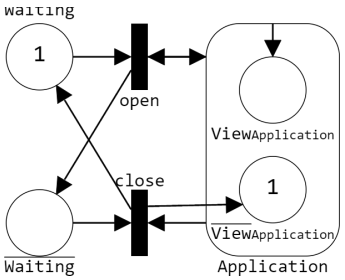
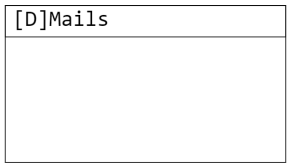


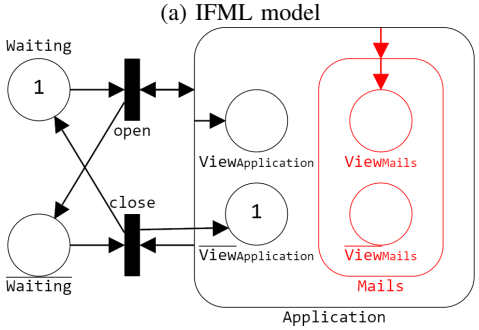
Fig. 3: PCN of an empty application

describing that a user can open the application, which is initially not in view. The open transition removes and adds a token from Application and moves a token from Waiting to $\overline{\text{Waiting}}$, disabling itself and enabling close. The close transition moves a token from Application to $\overline{\text{View}}_{\text{Application}}$ and moves a token from $\overline{\text{Waiting}}$ to Waiting, disabling itself and enabling open, thus resetting the application state to the default initial marking.

Figure 4a shows the second simplest scenario: an IFML model with one top-level default ViewContainer ($Mails$, in the example). This model corresponds to an application that shows a blank screen at start-up.



(a) IFML model



(b) Place Chart Net

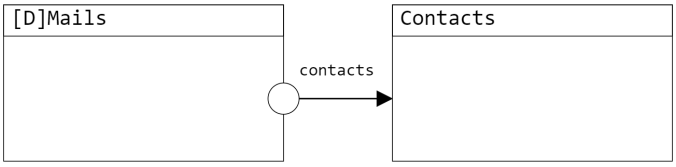
Fig. 4: Single, empty default ViewContainer Model

Figure 4b shows its mapping[1]: the PCN of Figure 3 is extended by introducing a place chart called Mails, child of Application, which is initialized by the parent. The Mails place chart has two children bottom place charts ($\text{View}_{\text{Mails}}$, initialized by the parent, and $\overline{\text{View}}_{\text{Mails}}$). The two $\text{View}_{\text{Mails}}$ states represent whether the ViewContainer is, or is not, in view. The firing of the open transition now adds also a token to $\text{View}_{\text{Mails}}$, meaning that the ViewContainer is displayed.

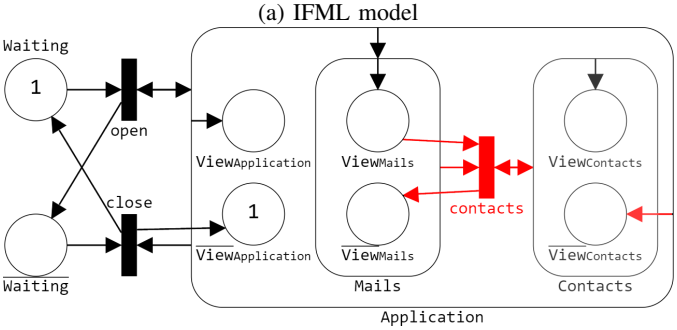
Navigation between ViewContainers. Figure 5a shows the elementary navigation step between top-level ViewContainers.

[1]In the following, the red color identifies the portions of a PCN affected (inserted or updated) by the mapping rule that the example illustrates.

The IFML model comprises two top-level ViewContainers ($Mails$ and $Contacts$), an Event called $contacts$ associated with the $Mails$ ViewContainer, and a NavigationFlow from such event, targeting the $Contacts$ ViewContainer.



(a) IFML model



(b) Place Chart Net

Fig. 5: Navigation between top-level ViewContainers

Figure 5b shows the PCN that maps the IFML model of Figure 5a. The Application top place chart contains two place charts Mails and Contacts, each one with two children bottom place charts ($\mathsf{View}_{\mathsf{Mails}}$, $\overline{\mathsf{View}}_{\mathsf{Mails}}$, $\mathsf{View}_{\mathsf{Contacts}}$, $\overline{\mathsf{View}}_{\mathsf{Contacts}}$). Mails is initialized by Application, because $Mails$ is the *default* top-level ViewContainer[2]; this causes the initialization by default of the $\mathsf{View}_{\mathsf{Mails}}$ place chart. Conversely, Contacts is not initialized as the default, but an initialization arc from Application targets the $\overline{\mathsf{View}}_{\mathsf{Contacts}}$ place chart, denoting that the $Contacts$ ViewContainer is not in view initially. The navigation between the two ViewContainers is represented by a transition named contacts, which denotes the change of the display status of the two ViewContainers. The transition moves a token from Mails, $\mathsf{View}_{\mathsf{Mails}}$ and Contacts to Contacts and $\overline{\mathsf{View}}_{\mathsf{Mails}}$. Note that the apparently redundant input place charts of the contacts transition (Mails, $\mathsf{View}_{\mathsf{Mails}}$) are necessary to ensure that: 1) tokens are consumed also for the possible children place charts of Mails; 2) the transition is enabled only when the $Mails$ ViewContainer is in view.

The next examples show the mapping of models that include ViewComponents, computed and rendered in the interface, possibly based on the value of some input parameters. The behavior of a ViewComponent can be regarded as the result of the interplay between two parts: 1) the *model*, representing the status of the interaction with the data source providing content to the ViewComponent; 2) the *view model*, representing the display of content in the interface. For example, in a pure HTML Web application, the *model* could be the data bean holding objects extracted from a database and the *view model* could be the HTML rendition of such objects. In an Android app, the *model* could be a Java object and the *view model* the GUI widget bound to it. The *model* part of a ViewComponent can be modeled by the following states: *Clear*: if the ViewComponent needs input to be computed (e.g., parameters appearing in its conditional expression); in this state, it cannot show any content and remains empty. *Ready*: this happens in two cases: the ViewComponent does not require any input parameter or it has received the needed inputs. *Computed*: if its content is computed and is therefore available to be displayed. The changes between these states are modeled by PCN transitions: *Propagate*: brings from *Clear* to *Ready* and represents the propagation of input parameters to the ViewComponent; *Compute*: brings from *Ready* to *Computed* and represents the computation of content using the received inputs; the content becomes available to the view model. The four different states of a ViewComponent are represented with four PCN bottom place charts: In, $\overline{\mathsf{In}}$, Out and $\overline{\mathsf{Out}}$ (for an example see Figure 6c). In and $\overline{\mathsf{In}}$ encode the availability of all the input data necessary for the computation of the ViewComponent; Out and $\overline{\mathsf{Out}}$ describe the completion of the computation, which makes the content available to the view model. The three possible states of the model are therefore represented by the following configurations:

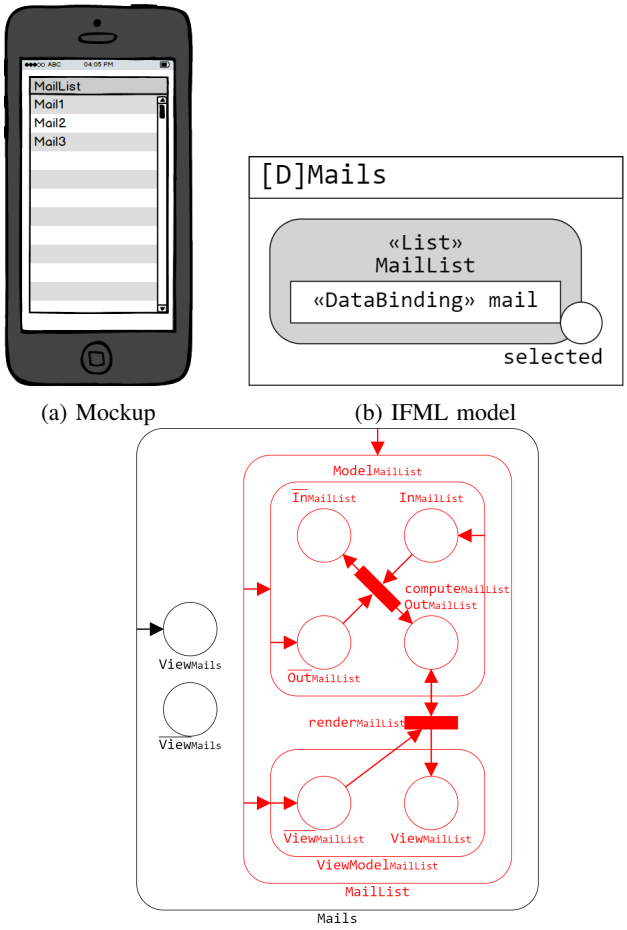
- *Clear*: a token in $\overline{\mathsf{In}}$ and a token in $\overline{\mathsf{Out}}$;
- *Ready*: a token in In and a token in $\overline{\mathsf{Out}}$;
- *Computed*: a token in $\overline{\mathsf{In}}$ and a token in $\overline{\mathsf{Out}}$.

The fourth configuration (a token in both In and Out) is not meaningful, since it represents the case where the content has been produced, but the inputs necessary for such computation have not been used. Figure 6c shows the *Compute* transition, which is a transition internal to the ViewComponent. The *Propagate* transition (not exemplified for space reasons) is commanded by an event external to the ViewComponent, e.g., the SelectCategory event between the CategoryList and ProducList ViewComponents in Figure 1b.

The *view model* part of a ViewComponent can be represented by two states: *Invalid*, denotes that the ViewComponent is not displayed. *Visible*, denotes that the ViewComponent has received data from the model and therefore can be displayed. These two states are represented by two bottom place charts View and $\overline{\mathsf{View}}$, respectively, as exemplified in Figure 6c. The change from the *Invalid* to the *Visible* state is modeled by the *Rendering* transition, shown in Figure 6c: it represents the copy of the content from the model to the view model and the consequent rendering of the ViewComponent. As an example, figures 6a and 6b present a simple IFML model with a top-level ViewContainer comprising only one ViewComponent: an application that displays a list of mails. Figure 6c shows the PCN mapping of the IFML model[3]. A place chart, child of Mails, named MailList represents the $MailList$ ViewComponent, initialized from the parent. It contains two child place charts $\mathsf{Model}_{\mathsf{MailList}}$ and $\mathsf{ViewModel}_{\mathsf{MailList}}$ representing the

[2]In IFML; the default ViewContainer is the one displayed when the parent container, or the whole application, is accessed.

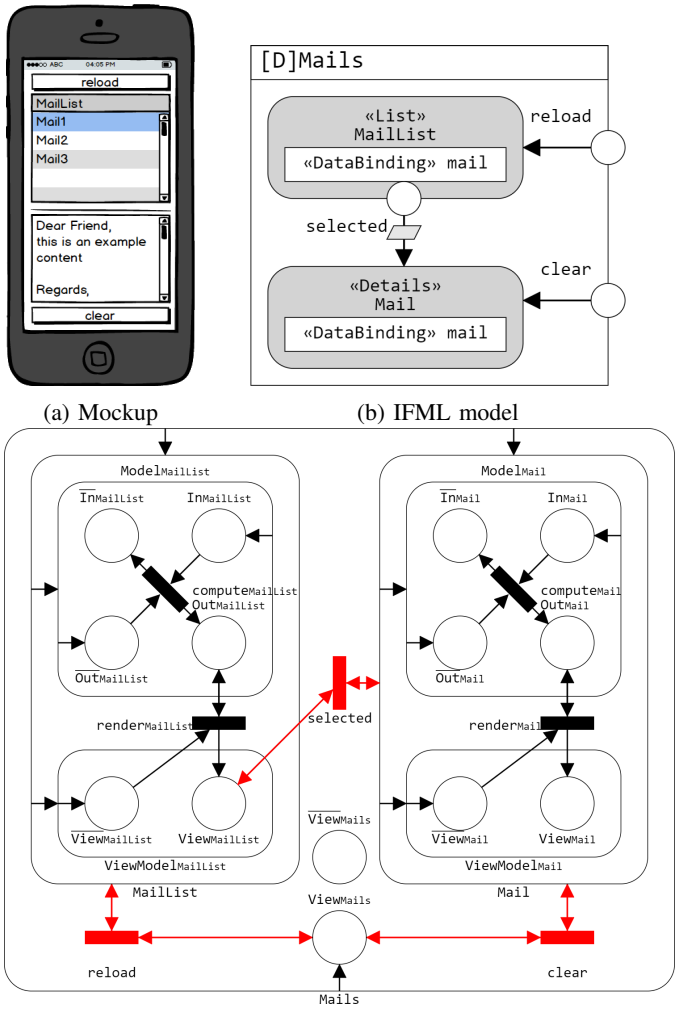
[3]From now on, for brevity we omit the Application top place chart and the $Waiting$ Boolean variable from the PCN mapping of the IFML models.



(a) Mockup (b) IFML model

(c) Place Chart Net

Fig. 6: Single ViewComponent model



(a) Mockup (b) IFML model

(c) Place Chart Net

Fig. 7: Navigation betweenViewComponents: navigation flow

model and the view model part of the ViewComponent, respectively. $\text{Model}_{\text{MailList}}$ contains the four bottom place charts defined earlier ($\text{In}_{\text{MailList}}$, $\overline{\text{In}}_{\text{MailList}}$, $\text{Out}_{\text{MailList}}$ and $\overline{\text{Out}}_{\text{MailList}}$). Transition $\text{compute}_{\text{MailList}}$ represents the computation of the content: it removes a token from $\text{In}_{\text{MailList}}$ and adds a token to $\overline{\text{In}}_{\text{MailList}}$, denoting the consumption of the inputs; it also removes a token from $\overline{\text{Out}}_{\text{MailList}}$ and adds a token to $\text{Out}_{\text{MailList}}$, denoting the availability of the model content.

$\text{ViewModel}_{\text{MailList}}$ contains the two bottom place charts $\text{View}_{\text{MailList}}$ and $\text{View}_{\text{MailList}}$. MailList also contains a transition describing the rendering of the view model, named $\text{render}_{\text{MailList}}$, which removes a token from $\overline{\text{View}}_{\text{MailList}}$ and $\text{Out}_{\text{MailList}}$ and adds one token to $\text{View}_{\text{MailList}}$ and $\text{Out}_{\text{MailList}}$.

The *MailList* ViewComponent is initialized by default from the *Mails* ViewContainer as follows: the model is set to the ready state and the view model to the invalid state.

Events and Navigation Flows. Figure 7 extends the previous example with events and navigation flows: the mail list is interactive, enabling message selection from the list and display in another interface component. When the application starts, only the list is displayed; after the user selects one item from the list, its details are shown. The *reload* event allows the user to refresh the list of mails; The *clear* event enables flushing the *Mail* message. visualization. Figure 7b illustrates the corresponding IFML model: a NavigationFlow from *MailList* to *Mail* has a parameter binding that associates the currently selected mail in the list with an input parameter in the filter condition of the *Mail* ViewComponent[4]. Each time the user chooses a mail message, the *selected* event is fired, the ID of the chosen message is made available as the new value of the input parameter of the *Mail* ViewComponent, and the new model content is computed based on such value; then, the view model part of the *Mail* ViewComponent is updated and displays the newly selected message. The PCN of Figure 7c illustrates the mapping of the IFML diagram of Figure 7b. Now the place chart of the *Mails* ViewContainer comprises two children place charts, corresponding to the *MailList* and *Mail* ViewComponents, (in particular, they are initialized to the ready state). Note that although the place chart associated

[4]For brevity, in the IFML diagram we omit the SelectorCondition of ViewComponents and the parameters in the ParameterBinding of the InteractionFlows, described in the text.

with the $Mail$ ViewComponent is initialized to the ready state, its input parameter initially has a null value because the user has not selected a message yet; thus, the computation and rendering transitions fire but the view model displays an empty ("null") content. The navigation flow from the $selected$ event maps into a transition ($selected$) that affects the $Mail$ ViewComponent as follows: the model place chart resets to the ready state and the view model place chart resets to the invalid state; the computation and rendering transitions fire, causing the refresh of the model content, based on the new value of the input parameter (the message selected by the user) and the display of the view model. The $reload$ and $clear$ events do not have any parameter binding, but their behavior is very similar: they reset the model place chart to the ready state and the view model place chart to the invalid state: as an effect, the previous content is invalidated and the model and view model are recomputed. Note that the NavigationFlow of the $clear$ event does not provide any input to the $Mail$ ViewComponent and thus the "null" content is displayed, with the effect of clearing the message interface area.
We refer the reader to IFMLEdit.org for more examples[5].

V. Implementation

The described mapping is implemented in IFMLEdit.org, where developers can create the IFML model in the online editor, then (optionally) map the model into a PCN and simulate it to study its dynamics; next, they can generate the code for the web or for a cross-platform mobile language, execute and validate the prototype, and turn it into a real app, by customizing look&feel and replacing mock-up data access with real calls. IFMLEdit.org exploits client-side editors and code generators developed in JavaScript, which, once loaded, can be used also offline. Developers new to the platform can start using it without installations and configurations, and also work offline and maintain full control and privacy of their projects. The system comprises four components: 1) *Model editor*: a visual editor to insert (by means of drag&drop) elements in the model, edit their property and connections (Figure 8); 2) *Model-to-JSON transformation framework:* a rule-based JavaScript engine that, given as input a model object, can generate an arbitrary output JSON structure. 3) *Browser-Server emulator:* 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. 4) *Mobile emulator:* a JavaScript component able to emulate a mobile cross-platform environment (now Cordova); it supports the execution of the generated cross-platform mobile code. Once the structure of the application is modeled, the developer can use the property editor to specify how ViewComponents connect between them and to the data sources. The latter aspect is specified by defining: *Collections*, which define sample Domain Model entities used as data-sources; *Filters:* which define entity attributes and predicates used to select relevant instances; the origin of parameters used to evaluate filters at runtime is specified by editing the IFML DataBinding associated with inbound NavigationFlows; *Fields:* define the entity attributes displayed in the UI and exposed as output by outbound NavigationFlows.

The developer can generate a formal description of the application by running the model-to-model transformation from IFML to PCNs. The PCN dynamics 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. Figure 9 illustrates the PCN model generated from the IFML diagram of Figure 8; the PCN simulation helps the developer identify inconsistencies in the specified application, such as unreachable states and race conditions.

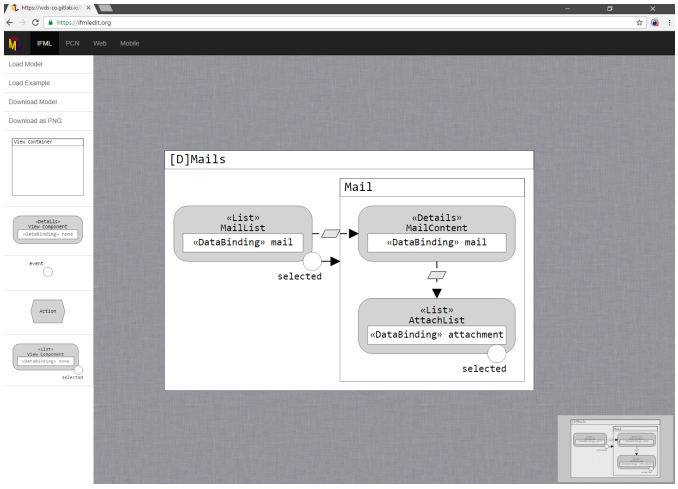


Fig. 8: Model editing

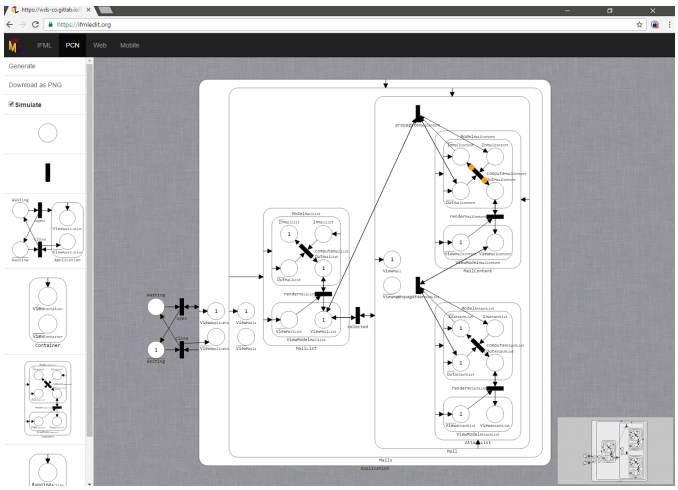


Fig. 9: Model semantics simulation

The developer can generate a fully functional web and mobile prototype, via model-to-code transformation. Figure 10 shows the generated web prototype, running in the web server emulated inside the browser. Figure 11 shows the generated mobile prototype, in the mobile emulator inside

[5]A Load Example button in the Editor allows the user to import models from a gallery.

the browser. In-browser emulation allows the developer to test the current web or mobile release without installing any web server and also in absence of the internet connection.

The prototype uses automatically generated sample data,

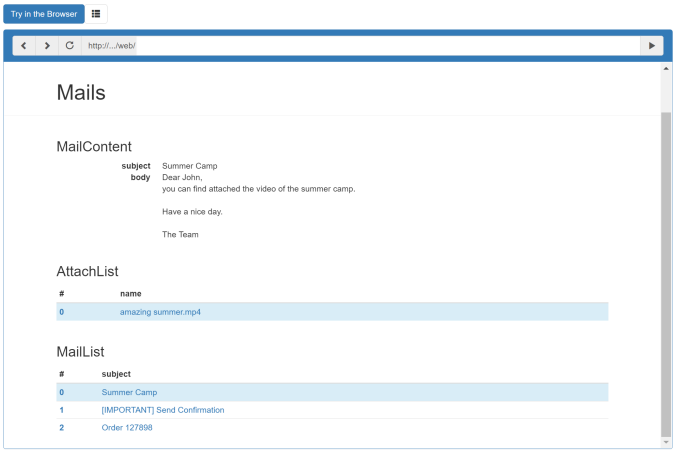


Fig. 10: Web code generation

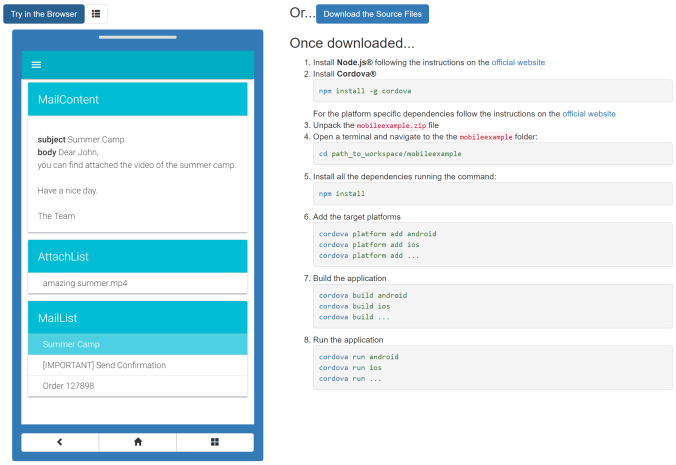


Fig. 11: Mobile code generation

so to populate the UI, allowing the developer to test the interface without loading content. For more realistic tests, the developer can edit, add or remove entities from the automatic database. IFMLEdit.org can be used repeatedly to evaluate different application structures (e.g., single vs multiple pages) and interaction approaches. The generated prototype can be downloaded and refined to produce the final application.

VI. Conclusions

We have described a MDD approach for the automatic generation of cross-platform web and mobile applications from IFML specifications. The semantics of IFML is defined by mapping it to PCN, a variant of Petri Nets, that can be used for model simulation. From the PCN and the platform specification (web or mobile) an executable application prototype can be delivered. The development process is supported by the online IFMLEdit.org tool. Future work will focus on extending code generation with advanced mobile design patters for online-offline usage, content caching and pre-fetching.

References

[1] "Appcelerator platform," http://www.appcelerator.com/, accessed: 2016-12-03.

[2] "IBM MobileFirst platform foundation," https://www.ibm.com/support/knowledgecenter/SSNJXP/welcome.html, accessed: 2016-12-03.

[3] "Adobe Phonegap," http://phonegap.com/, accessed: 2016-12-03.

[4] "Telerik appbuilder," http://www.telerik.com/platform/appbuilder, accessed: 2016-12-03.

[5] "Appery.io platform," https://appery.io/, accessed: 2016-12-03.

[6] "Codiqa builder," https://codiqa.com/, accessed: 2016-12-03.

[7] "Eachscape platform," https://eachscape.com/, accessed: 2016-12-03.

[8] "Alpha Anywhere platform," http://www.alphasoftware.com/, accessed: 2016-12-03.

[9] "AnyPresence platform," http://www.anypresence.com/, accessed: 2016-12-03.

[10] B. Selic, "The pragmatics of model-driven development," *IEEE Softw.*, vol. 20, no. 5, pp. 19–25, Sep. 2003.

[11] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*, 1st ed. Morgan & Claypool Publishers, 2012.

[12] L. Gaouar, A. Benamar, and F. T. Bendimerad, "Model driven approaches to cross platform mobile development," in *Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication*, ser. IPAC '15. New York, NY, USA: ACM, 2015, pp. 19:1–19:5.

[13] OMG, "Interaction flow modeling language (ifml), version 1.0," http://www.omg.org/spec/IFML/1.0/, 2015.

[14] M. Kishinevsky, J. Cortadella, A. Kondratyev, L. Lavagno, A. Taubin, and A. Yakovlev, "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*, ser. Lecture Notes in Computer Science, P. Azéma and G. Balbo, Eds., vol. 1248. Springer, 1997, pp. 328–347. [Online]. Available: http://dx.doi.org/10.1007/3-540-63139-9_44

[15] "Xtext platform," https://eclipse.org/Xtext/, accessed: 2016-12-03.

[16] "Applause project," http://applause.github.io/, accessed: 2016-12-03.

[17] "BuildUp app builder," http://www.buildup.io/, accessed: 2016-12-03.

[18] D. Steiner, C. Ţurlea, C. Culea, and S. Selinger, *Computer Aided Systems Theory - EUROCAST 2013: 14th International Conference, Las Palmas de Gran Canaria, Spain, February 10-15, 2013. Revised Selected Papers, Part II*. Springer Berlin Heidelberg, 2013, ch. Model-Driven Development of Cloud-Connected Mobile Applications Using DSLs with Xtext, pp. 409–416.

[19] H. Heitkötter, T. A. Majchrzak, and H. Kuchen, "Cross-platform model-driven development of mobile applications with md2," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ser. SAC '13. New York, NY, USA: ACM, 2013, pp. 526–533. [Online]. Available: http://doi.acm.org/10.1145/2480362.2480464

[20] C. Jones and X. Jia, "The AXIOM model framework - transforming requirements to native code for cross-platform mobile applications," in *ENASE 2014 - Proceedings of the 9th International Conference on Evaluation of Novel Approaches to Software Engineering, Lisbon, Portugal, 28-30 April, 2014*, 2014, pp. 26–37.

[21] C.-K. Diep, Q.-N. Tran, and M.-T. Tran, "Online model-driven ide to design guis for cross-platform mobile applications," in *Proceedings of the Fourth Symposium on Information and Communication Technology*, ser. SoICT '13. New York, NY, USA: ACM, 2013, pp. 294–300. [Online]. Available: http://doi.acm.org/10.1145/2542050.2542083

[22] S. Barnett, R. Vasa, and J. Grundy, "Bootstrapping mobile app development," in *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ser. ICSE '15. IEEE Press, 2015, pp. 657–660. [Online]. Available: http://dl.acm.org/citation.cfm?id=2819009.2819130

[23] "Mendix platform," https://www.mendix.com/, accessed: 2016-12-03.

[24] "Outsystems platform," https://www.outsystems.com/, accessed: 2016-12-03.

[25] "WebRatio platform," http://www.webratio.com/, accessed: 2016-12-03.

[26] OMG, "Business processs model and notation, version 2.0," http://www.bpmn.org/, 2011.