Jurnal Ilmu Komputer dan Informasi (Journal of Computer Science and Information)
17/2 (2024), 109-120. DOI: http://dx.doi.org/10.21609/jiki.v17i2.1178

Code Generator Development to Transform IFML (Interaction Flow Modelling Language) into a React-based User Interface

Ilma Ainur Rohma* and Ade Azurat†
Faculty of Computer Science, Universitas Indonesia
Depok, Indonesia
*Email : *ilma.ainur21@ui.ac.id, †ade@cs.ui.ac.id*

Abstract

Model-Driven Software Engineering (MDSE) is a software development approach that uses the Model to be the main actor of the development. MDSE can be applied to User Interface (UI) Development so that a model for the UI can be built, and then a transformation can be made to turn it into a running application. In this research, we develop UI Generator to support UI Development with the MDSE approach. This UI Generator can also support UI Development in Software Product Line Engineering (SPLE) paradigm. The UI is modeled with Interaction Flow Modeling Language (IFML) diagram. Then The IFML diagram is transformed into React-Based UI by the UI Generator. The UI Generator is developed with Acceleo on Eclipse IDE to transform IFML into React Code with the transformation rules defined in this research. The UI generator is also enriched with display settings and static page management to address user customization needs. The experimental results show that the UI Generator can generate a functional website. Besides evaluating the working product, UI Generator is evaluated qualitatively well based on six quality criteria as an SPLE supporting tool.

Keywords: *Model-Driven Software Engineering (MDSE); User Interface (UI) Development; Code Generator; Interaction Flow Modeling Language*

1. Introduction

One of the software development approaches is model-driven software engineering (MDSE). The basic idea of MDSE is that human thinking naturally starts from making an abstraction that includes generalization, classification, and aggregation [1]. In the Software Engineering area, making the abstraction is known as modeling.

Models have a vital role in MDSE, but the model is still an abstraction. The transformation in MDSE makes this abstraction a concrete application [1]. A code generator, a model-to-text (M2T) transformation tool, can be used to automatically transform the model into running code [2].

MDSE can be applied to User Interface (UI) development as part of software development. UI provides a graphical or textual form that the user can use to interact with the system [3]. To develop UI with the MDSE approach, we must make an abstraction of the UI [4, 5].

Interaction Flow Modelling Language (IFML) can be used to represent the UI as an abstract UI Model [3, 6–9]. IFML consists of elements representing the UI elements and UI behaviour [2]. Each element of IFML has a notation that can be combined to construct the abstraction on the UI. Based on Model-Driven Architecture, an MDSE in practice, IFML is in Platform-Independent Model (PIM) level [1]. Design and analysis of UI requirements happen at the PIM level so that the behavior and structure can be described at this level. [1].

This research proposes an MDSE approach to develop the user interface for a website. Sboui et al.[8] and Bernaschina et al. [9] develop UI Generators to transform a UI model into UI code. Sboui et al. [8] using Concrete User Interface (CUI) to model the UI and then transforms it into a UI code based on standard HTML and CSS. Bernaschina et al. [9] develops UI-Generator to transform IFML into Nodejs-based UI, but only a few IFML notations can be transformed in this research, and they do not consider how to customize the UI.

This research focuses on User Interface Development, so we developed a code generator called UI Generator that transforms IFML into User Interface code based on React. React is Javascript Library for

building web or native user interfaces [1]. We can build a more flexible and faster user interface with React than basic javascript [10].

This research starts with the process of modeling the UI with IFML. IFML must be properly constructed to represent a UI based on the standard IFML rules [11]. Then this research defines the transformation rules to specify the transformation of an element into a react code. Transformation rules are defined by using the various properties that IFML provides. The transformation rules are a crucial part of this research because we cannot develop a proper UI from IFML if the rules have an error. UI Generator is developed based on the transformation rules as a tool to prove that the transformation rules can be implemented.

After that, the evaluation phase is conducted to evaluate the research proposal. Quantitative evaluation is done by doing a functional test to get the percentage of features that are running well. This evaluation will prove the correctness of the transformation rules. Besides the functional test, Six quality criteria from Apel et al. [12] are used to evaluate the quality of the UI Generator qualitatively.

This UI Generator is also used in Software Product Line Engineering [4]. In SPLE, developing UI must be adaptable to match the application variation [3]. So with IFML, we can build models for all features and choose which feature will be chosen in a specific web. Then a user interface can be automatically generated using a UI generator.

This research is used to build a website for a charity organization with SPLE. After we have all possible features, IFML is created for each feature. Then the transformation rules are defined for each element in IFML into React Code. With those transformation rules, The UI Generator with Acceleo in Eclipse can be developed. After that, the UI Generator is evaluated by running it to generate a website from the IFML that was created before.

The following sections of this article are organized as follows: Section II presents the running example to give an overview of the tools. In Section III, UI Modelling with IFML is explained in detail. Section IV presents the transformation rules that we proposed for the UI generator. In Section V, we explain the development process of the UI generator. Section VI presents the evaluation of the UI generator and future works. Last, This article is concluded with the presentation of our works in section VII.

[1] https://react.dev/

2. Running example

In this reasearch, UI Generator used for developing a website for Charity Organization. The website has several feature like Program, Income and Expense record, donation, etc. In this paper, we show how UI Generator works to build Program Feature. Program feature is a Create Read Update and Delete (CRUD) feature that contains four pages.

The snippet of IFML Diagram that we use in this research is shown in Fig. 1. Fig. 1 is IFML Diagram for Activity Feature. Some of the components details are explained in the next section. This IFML is the input for our UI Generator and the output is UI code based on React. UI Generator produce four page based on in figure 1.

Fig 2 is a page for list of programs from Daftar Program page `ViewContainer`. This `ViewComponent` contains `List ViewComponent` in this example we show you a List as a Card Grid not Table. The detail button in Fig. 2 navigate us to Detail Page that show in Fig. 3. This detail page is from Detail Program Page `ViewContainer`.

The other two pages are the page that contains a form. The first is Fig. 4 is an Add Program Page from Tambah Program Page `ViewContainer` that contains a blank form. Different with Fig. 5, the form in this page have an intial value. Fig. 5 is from Ubah Program Page `ViewContainer` and it have a different form model with Tambah Program Page `ViewContainer`.

Activity IFML diagram in Fig. 1 is combined with various IFMLs for other features in a model. The model is in one .core file containing an XML representation of the diagram. The UI generator reads this file to produce a complete website.

3. UI Modelling

As we explain in the Introduction, an abstraction of UI is built by IFML Diagram. IFML Diagram has some notations representing the UI elements and website behaviour. Table 1 shows the list of IFML notations we use to construct the UI abstraction. The following sections explain how we construct IFML diagrams for making a UI in detail.

Website Page

A Website Page is represented by a `ViewContainer`. In `ViewContainer`, we can compose various elements that represent website content [11]. Three types of `ViewContainer` exist: XOR, Default, and Landmark.

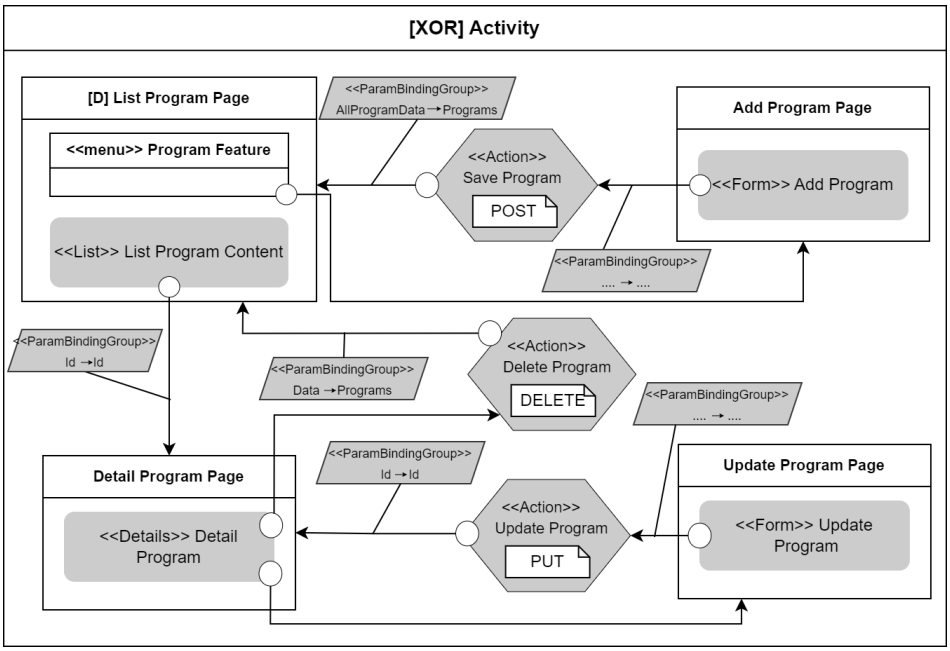


Figure 1. IFML Diagram for Program Feature

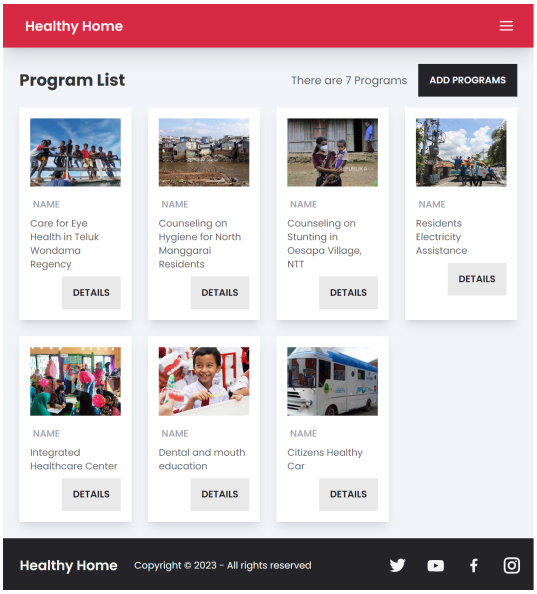


Figure 2. List of Programs Page

In the Running Example above, we use XOR, Default, and Standard `ViewContainer`. `XOR ViewContainer` contains all the website pages related to a feature. One `XOR ViewContainer` represents a program feature. In `XOR ViewContainer`, we must compose `Default ViewContainer`. This `Default ViewContainer` represents the first page of a feature. The other `ViewContainer` is just a standard page interacting with other `ViewContainer`. So, from the IFML Digaram in Fig. 1, we can have four website pages that we can access.

`ViewContainer` is also have extension called `Menu ViewContainer`. This `Menu ViewContainer` is used to create a submenu on a web page. In Fig. 1, we use `Menu ViewContainer` in "List Program Page" `ViewContainer` to make an additional button "Add Program" in Fig. 2.

Each `ViewContainer` has `ViewComponent` to represent the page's content. In IFML metamodel, both are subclasses of `ViewElement` [11]. `ViewComponent` has three extensions to represent website content. There are `List`, `Form`, and `Detail`.

List and Table

The first `ViewComponent` extension is `List`. This `ViewComponent` represents the page content presented in a List or Table. The displayed content data comes from `DataBinding`. `DataBinding` have a URI property that can store API endpoint to retrieve the data. Data from `DataBinding` is shown by `VisualizationAttributes` element. The IFML diagram for List can be shown in Fig. 6

Form

A form in a website page is represented by `Form ViewComponent`. The form fields can be represented

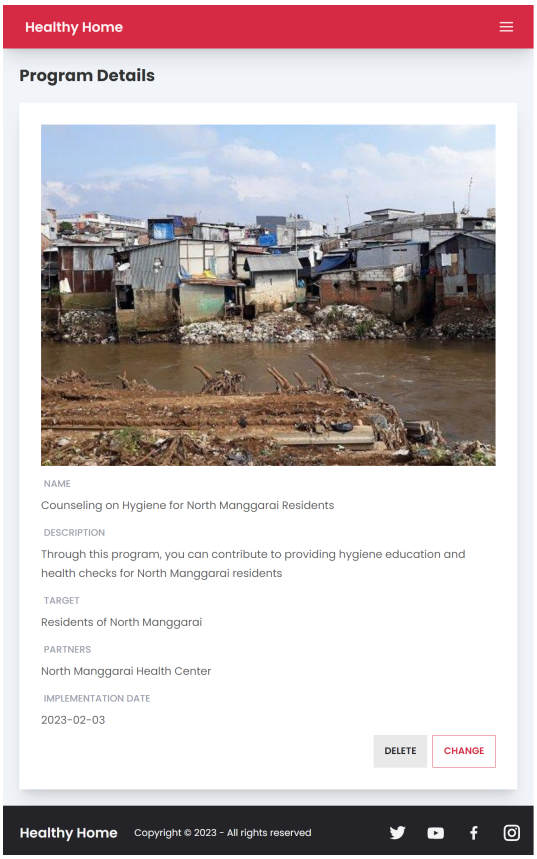


Figure 3. Detail of A Program Page

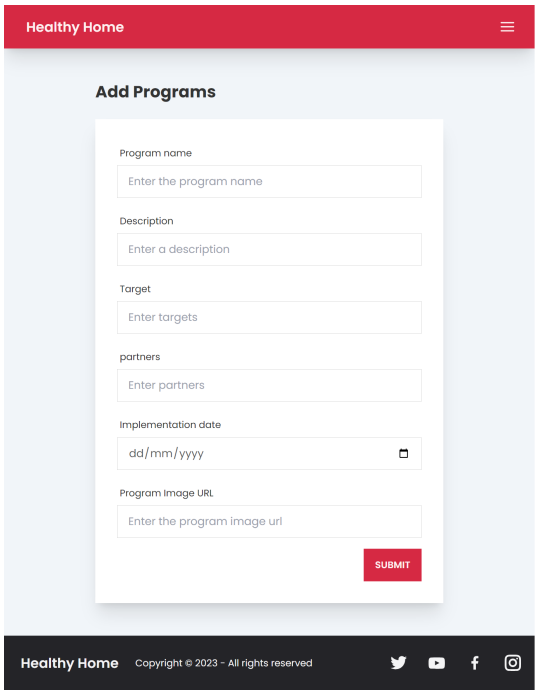


Figure 4. Form Add Program Page

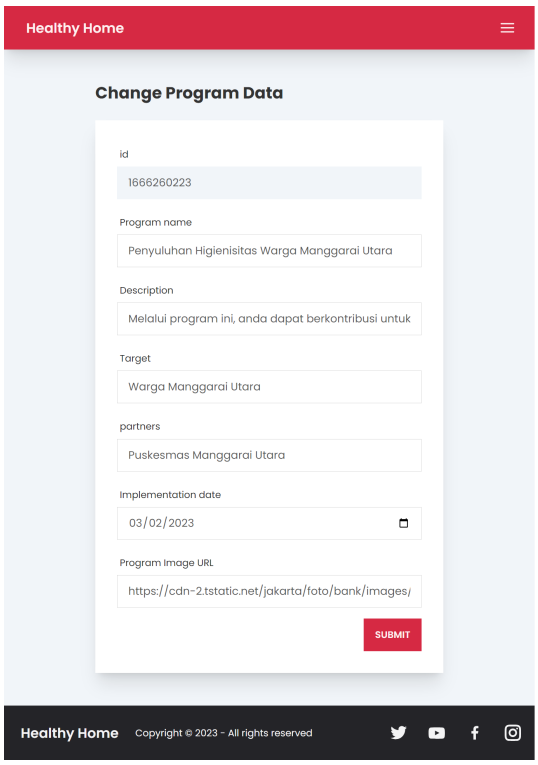


Figure 5. Form Update Program Page

by composing the `Field` element inside the `Form ViewComponent`. `Field` has two type: `SimpleField` and `SelectionField`. `SimpleField` represents a regular field and `SelectionField` represents drop-down list. A form for Add Page does not need initial data, so we can represent this just a `Form` in `ViewComponent` as Fig. 7 shows.

The initial data from the service is needed for the Update Page. In this case, all `Fields` are wrapped in `DataBinding`. The data from `DataBinding` is displayed using the `Slot` element inside the `Field`. Fig. 8 shows all `SimpleField` in Fig. 7 are wrapped in a `DataBinding` and each field has a `Slot`. Also, `DataBinding` in `Form` can be used to retrieve data for options in `SelectionField`.

Detail

`Detail` is one extension of `ViewComponent` that used to display details of a data [11]. Same with `List`, the displayed data is retrieved and shown by `DataBinding` and `VisualizationAttributes`. In this case, we use the two types of `Parameter` as

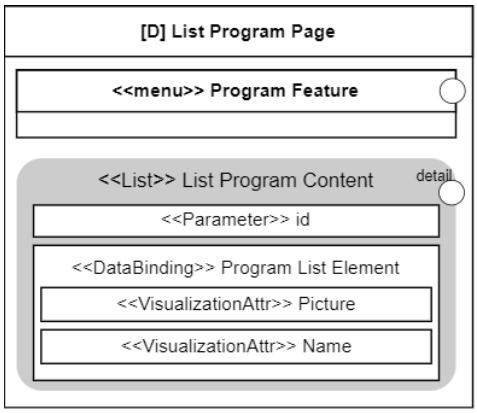


Figure 6. IFML Diagram for `List ViewComponent`

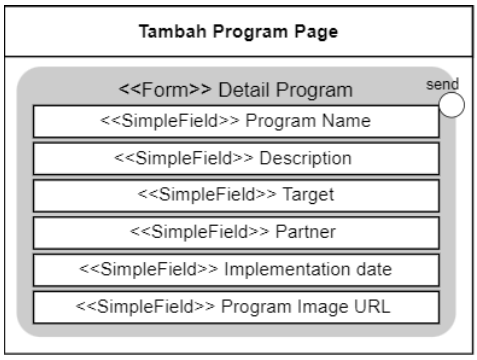


Figure 7. IFML Diagram for `Form ViewComponent` without Initial Data

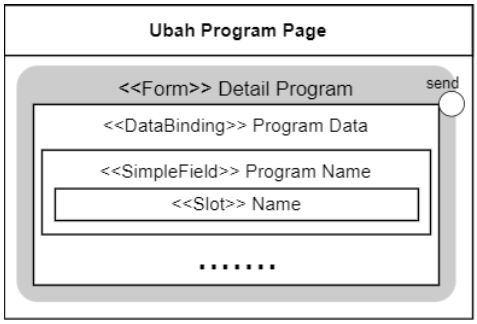


Figure 8. IFML Diagram for `Form ViewComponent` with Initial Data

Fig. 9 shows. First, the `Id Parameter` with direction `INOUT` is used to save the passed parameter from an event. Second, the `Object Detail Program Parameter` with direction `OUT` is used to retrieve data from the service.

Service

Doing a communication service between UI and database is important in website development.

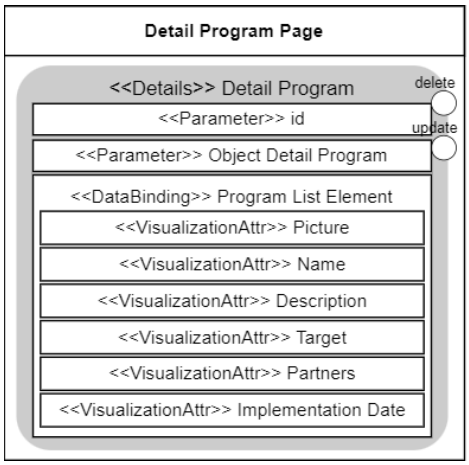


Figure 9. IFML Diagram for `Detail ViewComponent`

This service is represented by `Action` element in IFML Diagram [11]. In the running example, data is fetched and sent using a REST API. So, information on the HTTP method must be added in every API call. The `Annotation` element is used in `Action` to contain information about the HTTP method. `Action` can also have an `Event` called `Action Event`. This `Action Event` represents where the page is redirected after the call is finished.

Button and Navigation

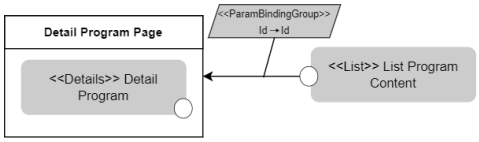


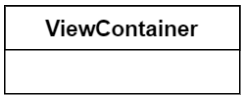
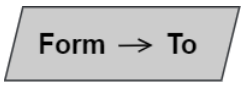
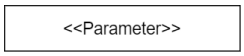
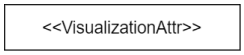
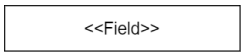
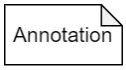
Figure 10. IFML Diagram for `ViewElementEvent` and `NavigationFlow`

The website has some button to trigger some event or redirect us to another page. This element and behavior are represented by `ViewElementEvent` and `NavigationFlow`. `ViewElementEvent` have a `NavigationFlow` to specicify the behaviour of the UI Event. When navigating, `NavigationFlow` can have a `ParameterBinding` to represent all parameters passed. Fig. 10 shows that if we trigger a button from `List`, the page will be redirected to the Detail Program Page by passing an id parameter as id.

4. Transformation Rules

Before developing UI Generator, each element in IFML is defined by its transformation rules. In this research, IFML is transformed into React based

Table 1. List of IFML elements

Notation Name	Description	Notation
`ViewContainer`	represents a website page	
`ViewComponent`	representa a website content	
`Action`	represents the API call	
`Event`	represents a Button	
`NavigationFlow`	represents a transition on the website	
`ParameterBinding`	represents the parameters carried in the navigation flow	
`Parameter`	represents the value stored on the web page	
`VisualizationAttributes`	represents an element that displays data	
`Field`	represents a field in the form	
`Annotation`	represents a note on a particular element	

User Interface. This following items describe how IFML Diagram in Running Example transformed into React code.

`ViewContainer`

`ViewContainer` is an element which represents a website page containing other UI elements. This element is transformed into a ReactJS component that referenced by routing. Fig. 11 shows that we transformed `ViewContainer` along with `ViewComponent` by using its `name` property.

`ViewComponent`

`ViewComponent` represents the UI element that displays website content or accepts input from a user. `ViewComponent` generally is transformed into

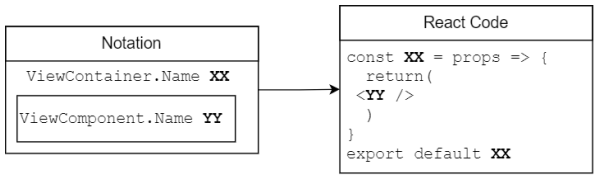


Figure 11. Transformation Rules for ViewContainer

a React Component that are composed into components from `ViewContainer`. `ViewComponent` has three extensions: `List`, `Details`, and `Form`. Each extension have a different component to store the content based on its form.

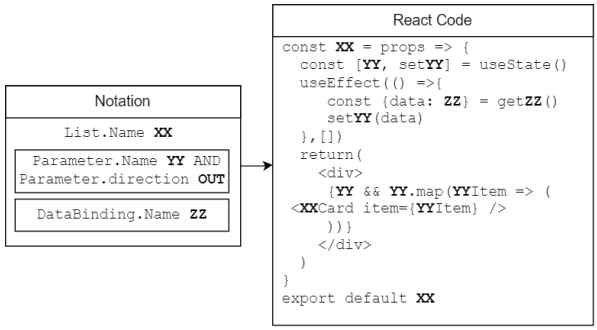


Figure 12. Transformation Rules for List

First we have `List ViewComponent`, the displayed data is represented by `Parameter` and `DataBinding`. The transformation rules of `List` is shown in Fig. 12. The `name` from `DataBinding` is used to call a service for retrieved data and save them to state from `Parameter`. UI generator provide two style of list, first is Card Grid View like we show in Fig. 2 and second is Table View. If the `name` property starts with "Table" the return value of the component is a table tag.

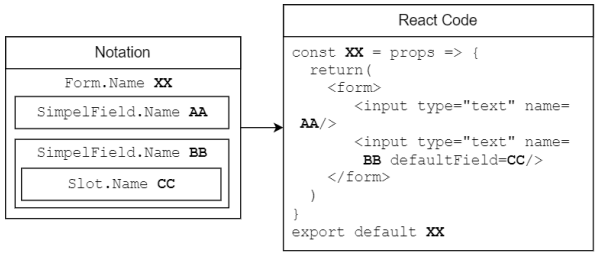


Figure 13. Transformation Rules for Form

The second extension of `ViewComponent` is Form. The transformation rules of `Form` is shown in Fig. 13. In form, some `SimpleField` or `SelectionField` represent fields on the website. `SimpleField` represents the common input field, and `SelectionField` represents a dropdown list field. `Slot` is used for the field that have a default value. So, the `name` property of `Slot` fill the `defaultField` of `<input>`

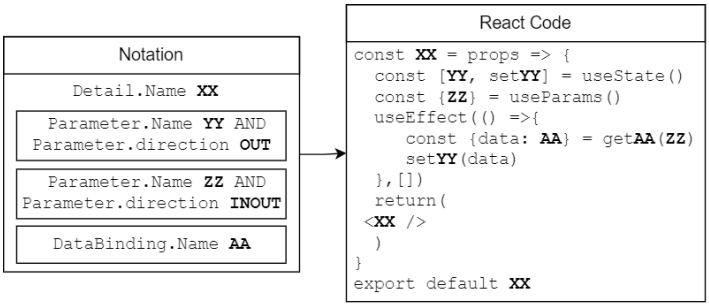


Figure 14. Transformation Rules for Detail

Fig. 14 shows the transformation rule of Detail. In `Details`, same with `List` the required data is represented by `Parameter` and `DataBinding`. `Out Parameter` is used for save the retrieved data from API. `Inout Parameter` is used to store parameters for service. The retrieved data from `DataBinding` is shown by the transformation of `VisualizationAttributes`.

`Event`

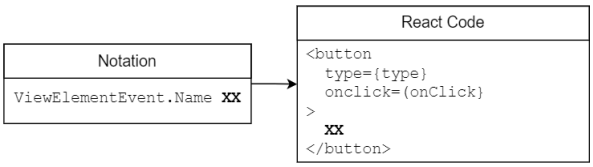


Figure 15. Transformation Rules for Event

`Event` represents an interaction by a user on a website to trigger certain functionality. This element is transformed into a button with an event handler as seen in Fig. 15. The main functionality of `ViewElementEvent` is calling the service or navigating to another page that may be passing a particular data. Therefore, to represent this functionality, `ViewElementEvent` has a `NavigationFlow` Element.

`NavigationFlow`

`NavigationFlow` represents the UI Changes when a certain event is executed. `NavigationFlow` can only target either ViewContainer or Action. Fig. 16 shows us transformation rules of

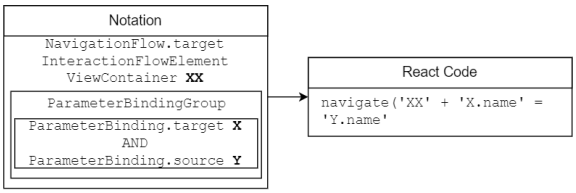


Figure 16. Transformation Rules for NavigationFlow Targeted ViewContainer

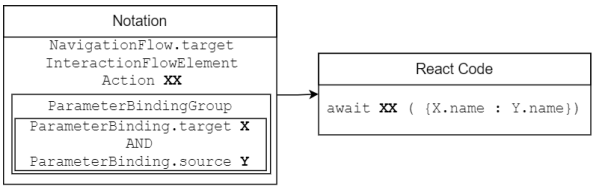


Figure 17. Transformation Rules for NavigationFlow targetted Action

NavigationFlow targeted ViewContainer. The transformation rules for NavigationFlow targeted Action shows in Fig. 17. Inside the NavigationFlow we can put ParameterBindingGroup that contains at least one of ParameterBinding. This ParameterBinding becomes a parameter that is carried over when navigating to another page or when calling aservice. The other NavigationFlow targeting Action will call service by executing a call() function in service generated from Action element.

Action

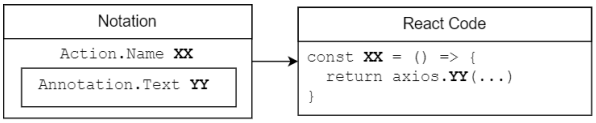


Figure 18. Transformation Rules for Action

Action describes the UI's business process triggered by an event. This element represents an HTTP request to API to have data transactions from the backend. In ReactJS, Action is transformed into a service that has its directory. This service makes an HTTP request using the Axios library like we show in Fig. 18. The name property from Action is used to find the HTTP method and backend endpoint. The HTTP methods for API calling can we use text from annotation in the Action element. The response from API is returned in JSON format that UI can read.

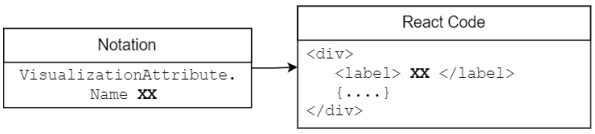


Figure 19. Transformation Rules for VisualizationAttributes

VisualizationAttributes

VisualizationAttributes represents a UI element that displays specific data. Usually, VisualizationAttributes is used to display data in List or Details. This element is transformed into an HTML container that contains a label and the particular data as seen in Fig. 19. The content label comes from the Name property and the particular data comes from FeatureConcept property.

DataBinding

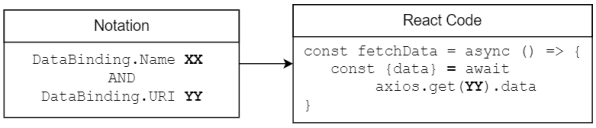


Figure 20. Transformation Rules for DataBinding

IFML diagram has Domain Model to represents instance of an element that can we bind to the system with DataBinding [11]. DataBinding has URI property to store API endpoint for retrieving the data. This URI is called asyncronously by axios with GET method to retrieved the data. The transformation rule for DataBinding can be seen in Fig. 20

5. UI Generator Development

Based on those transformation rules, a UI generator to transform the IFML diagram to React is developed using Acceleo. Acceleo is a model-to-text transformation tool based on Eclipse IDE. Acceleo uses a template-based approach to produce code from a model with Model to Text Language (MTL)[13] that is automatically executed by Java. Acceleo is equipped with various Object Constraint Language (OCL) functions that make it easy to process the input models. In addition, Acceleo can invoke Java code if we need some additional functions in UI Generator.

Acceleo has some modules in .mtl files that can contain templates and queries. A template defines sets of statements to generate the code file, and a

query extracts specific information from the model. Among these modules is the main module, the first executed module, which describes the UI generator's generation workflow. Also, additional services can be obtained by generating a query module from Java code to retrieve specific information from a model.

Before we developed the UI generator with Acceleo, A UI was developed using ReactJS to guide us in creating a template. From that code artifact, we trace where transformation rules are implemented to that code and make an mtl file. First, all the `ViewContainer` and `Action` elements are read by the UI Generator. From `ViewContainer`, UI Generator can read the `ViewComponent`, and different modules are done for each `ViewComponent`. Inside the `ViewComponent` modules, all components in our IFML Diagram can be read and transformed.

This UI Generator from Acceleo can produce a basic React-based User Interface. Every IFML input for this UI generator is transformed into the same style website. In reality, each company or person needs a unique personal website. So, we have to add a tool to maintain a variety of website styles and manage the website's static page.

GrapesJS [2], a web builder framework, is used for providing static page management. GrapesJS allows users to customize the page using a drag-and-drop visual editor. GrapeJS produce HTML and CSS files from a visual editor. These files are combined in the ReactJS template used for UI Generator.

Website product from UI Generator uses tailwindCSS [3] for styling Framework. Users are provided with the ability to manage website style through the creation of interface kits for display settings. This interface kit allows the user to choose a color theme and some competent style for a website. The interface kit is developed using DaisyUI [4]. DaisyUI is a UI Library to define every UI component in Tailwind. Just like GrapesJS, DaisyUI is also combined in the ReactJS template.

6. Evaluation

This research is evaluated with two aspects; generated website and UI Generator. First, the generated website is evaluated using Functional Testing. If the generated website can satisfy all scenarios, the the transformation rules are correct. Second, the UI Generator is evaluated qualitatively based on six quality criteria [12]. The six quality criteria are constructed to evaluate how well the tools can support SPLE.

Scenarios for all features based on website requirements are constructed for Functional testing. Researchers run all scenarios on the website; if the expected results are met, the scenario is passed. This following items are list of feature that we test on Website.

- Authentication and Authorization : user can Sign Up, user can Login via username and password, user can Login via Google, user can Logout.
- Activity or Program Feature : user can create, edit and delete a program, user can read list of programs, user can read detail of a program.
- Income Feature : user can create, edit and delete an income, user can read list of incomes, user can read detail of an income.
- Expense Feature : user can create, edit and delete an expense, user can read list of expenses, user can read detail of an expense.
- Static Page Management : user can view about us static page, user can update the information in static page.
- Display Settings : user can choose one of color theme and display style, user can view the preview.

From this Functional testing, 100% of tests are a success. With this result, we can conclude that the UI generator successfully produces a website.

In addition to the website product evaluation, the UI Generator itself is evaluated qualitatively. As explained in Section 1, this UI Generator can be used for Software Product Line Engineering (SPLE). So, this research use six quality criteria to evaluate UI Generator as a tools for SPLE [12]. This following items describe the evaluations of UI Generator based on the criteria.

- Preplanning Effort
  Sometimes, the user wants to change any functionality in this feature, like adding some input file, changing the button navigation, or adding another page. In another case, users want to add features to their website as the organization grows. Users can deal with those cases by updating or adding the IFML diagram. They do not need to deal with the code that must have technical knowledge. We can conclude that UI Generator have Low Preplanning Effort to develop a UI.
- Feature Traceability
  Feature traceability can be fulfilled if the UI Generator can be traced for every feature. For Example, if user wants to edit the form in a specific page, they can trace the IFML

[2]https://grapesjs.com/
[3]https://tailwindcss.com/
[4]https://daisyui.com/

Diagram from the `ViewContainer` and they can have the `Table ViewComponent`. We produce code that conforms to the IFML diagrams, which makes separate folders based on the feature in the diagram. With this approach, a developer can trace the code artifact easily to find which file wants to be improved.

- Separation of Concerns
  Similar to the analysis on feature traceability, each feature has its folder, and each element has its file. In other words, UI Generator will generate a feature directory that can serve its functionality independently from other feature directories. Model and UI generator development are done separately in the UI development process, so they do not affect each other. So this UI generator fulfill the separation of concern criteria.
- Information Hiding
  Information Hiding means that each feature has its internal and external parts. One application of this criteria is to develop features for static page management and display settings. Static page management uses a module from GrapeJS, so it has an internal part to compose the drag-and-drop visual editor and pass the HTML and CSS code to an external part. Like Static page management, display settings have internal parts of developing the interface kit and external parts of passing the styling parameters. So another module can use its external part without knowing how it works in internal parts.
- Granularity
  For granularity, the systems can be defined as fine-grained and coarse-grained when they have system changes. Fine-grained means the changes are at the lower level, and coarse-grained means the changes are at the top level. In this UI Generator, when user want to edit or add a new statement or add a new variation of the interface kit for UI, they can update the javascript file on ReactJS Templates. Because the javascript files are lower, It is classified as fine-grained. But if there are changes in transformation rules, they must add some files or edit some modules in the UI Generator code. These changes may affect other modules, so this can be classified as coarse-grained.
- Uniformity
  For UI development with UI Generator, developer have to start with making an IFML diagram for the input of UI Generator. Similarly, another developer can create an IFML diagram for a new application or feature because IFML has a standard rule. Another example is when UI Generator transforms IFML to ReactJS; all features have a uniform file structure. So, this UI generator satisfies the criteria because we have a similar manner to developing a product.

From functional testing and six quality criteria analysis, the UI Generator is good enough to develop UI. UI Generator also supports Model-driven Software engineering. We develop UI based on IFML diagram as a model and automatically transform it to Running UI code.

In this research, transformation rules for all components in IFML have not been defined IFML has 58 elements in the core package and ten in the extensions package based on IFML Meta Model. We use some of those in our works to define the transformation rules based on the used element. Transformation rules are defined for 44.16% of all elements in this research. If the user adds elements that do not have transformation rules, UI Generator cannot process them. For further work, we must define all the IFML Meta Model elements.

Reliable software must have a method to handle the failure [14]. So the UI Generator is still improving the ability to inform the user if there are failures when the UI Generator is running. The most likely failure is an error while modeling the UI that causes UI Generator to keep processing it. If it happens, the user needs to have the technical skill to fix it.

7. Related Works

In the previous research, Bernaschina et al.[9] developed UI Generator (IFMLEdit.org) to generate UI for website and mobile application using IFML and transform it to Nodejs-based UI using ALMOsT.js Framework [15]. Bernaschina et al.[9] mapped IFML elements to JSON representation and transformed them into UI Code. Bernaschina et al.[9] not transform all IFML element; they just transform `ViewContainer`, `ViewComponent`, `Event`, `Action`, and `NavigationFlow`. In our research, UI Generator is developed to transform all elements in standard IFML and directly transforms to the UI without creating a new representation.

UI Generator for a website is also developed by Yigitbas et al.[7]. Yigitbas et al.[7] combine IFML with ContextML and AdaptML to produce a Self-Adaptive User Interface (SAUI) that can adapt in the runtime based on context-of-use. Yigitbas et al.[7] focuses on integrating the IFML-UI generator with the adaption service but does not specify how IFML

is transformed into a UI code. This research focuses on transforming each element in IFML inti UI code by constructing transformation rules and developing UI Generator.

Sboui et al.[8] use the Software Product Line (DSPL) approach to build Context-adaptable User Interface. Sboui et al.[8] use Concrete User Interface (CUI) Model to make an abstraction for the UI. Unlike IFML, CUI Model cannot model specific elements in UI. The difference between Sboui et al.[8] and our work is the usage of SPLE concept and the modelling language used. This research develops a UI Generator that can support the SPLE to build UI, and we use IFML to make an abstraction of a UI.

8. Conclusion

Model-driven Software Engineering (MDSE) is one of the UI development approaches. In the MDSE model and transformation are essential to developing the UI. Interaction Flow Modeling Language (IFML) is one of the platforms that can model UI. IFML can make an abstraction of the whole UI, like all page content and interaction between content. UI Generator is developed to transform IFML into React Code. The generated React Code is enriched with display settings and static page management by UI Generator. Then the React Code artifact can be run as a Website User Interface.

The generated UI is evaluated by functional testing, so we can conclude that the UI Generator can produce the running website UI. Regarding quality, the UI Generator is evaluated with six quality criteria, and we know that the UI Generator can be an SPLE supporting tool. The UI Generator can transform 44.16% of IFML elements to React Code, so transformation rules have to be created for other IFML Elements.

Acknowledgment

This research was supported by Cornelita, Asfiolitha, Alisha, and fellow members at the Reliable Software Engineering (RSE) Laboratory, Faculty of Computer Science, Universitas Indonesia.

References

[1] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in practice*. San Rafael, United States: Morgan Claypool, 2017.

[2] M. Brambilla and P. Fraternali, *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML*, 12 2014.

[3] H. S. Fadhlillah, D. Adianto, A. Azurat, and S. I. Sakinah, "Generating adaptable user interface in sple: Using delta-oriented programming and interaction flow modeling language," in *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 2*, ser. SPLC '18. New York, NY, USA: ACM, 2018, p. 52–55.

[4] A. Pleuss, B. Hauptmann, D. Dhungana, and G. Botterweck, "User interface engineering for software product lines: The dilemma between automation and usability," in *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, ser. EICS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 25–34.

[5] A. Pleuss, S. Wollny, and G. Botterweck, "Model-driven development and evolution of customized user interfaces," in *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, ser. EICS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 13–22.

[6] A. Sajji, Y. Rhazali, and Y. Hadi, "An approach to automate generation of graphical user interfaces through ifml," vol. 201, 2022, p. 621–626.

[7] E. Yigitbas, I. Jovanovikj, K. Biermeier, S. Sauer, and G. Engels, "Integrated model-driven development of self-adaptive user interfaces," vol. 19, no. 5, 2020, p. 1057–1081.

[8] T. Sboui, M. Ben Ayed, and A. M. Alimi, "A ui-dspl approach for the development of context-adaptable user interfaces," vol. 6, 2018, p. 7066–7081.

[9] C. Bernaschina, S. Comai, and P. Fraternali, "Online model editing, simulation and code generation for web and mobile applications," in *Proceedings of the 9th International Workshop on Modelling in Software Engineering*, ser. MISE '17. IEEE Press, 2017, p. 33–39.

[10] P. Hunt. React: Rethinking best practices. JSConf-Youtube. [Online]. Available: https://www.youtube.com/watch?v=x7cQ3mrcKaY

[11] *Interaction Flow Modeling Language*, Object Management Group, 2015. [Online]. Available: http://www.omg.org/spec/IFML/1.0/

[12] S. Apel, D. Batory, C. Kästner, and G. Saake, *Feature-Oriented Software Product Lines*. Berlin: Springer-Verlag, 2013.

[13] "Acceleo: Overview," 2021. [Online]. Available: https:

//www.eclipse.org/acceleo/overview.html

[14] I. Sommerville, *Software engineering, Tenth Edition*. Pearson Education, 2019.

[15] C. Bernaschina, "Almost.js: An agile model to model and model to text transformation framework," in *Web Engineering*, J. Cabot, R. De Virgilio, and R. Torlone, Eds. Cham: Springer International Publishing, 2017, pp. 79–97.