From Interaction Flow Modeling Language to Symfony: an automated model transformation methodology

Abir Sajji, Yassine Rhazali, Youssef Hadi
RSI Laboratory, Higher School of Technology, Ibn Tofail University, Kenitra, Morocco

Article Info

Article history:

Received Jan 12, 2025
Revised Aug 17, 2025
Accepted Sep 1, 2025

Keywords:

Atlas Transformation Language
Interaction Flow Modeling Language
Model transformation
Model-driven architecture
Platform independent model
Platform specific model
Symfony

ABSTRACT

Web application development has become increasingly complex with the rise of modern frameworks and user-centric architectures. Ensuring efficient and reliable development processes requires adopting structured methodologies that bridge abstract models with platform specific implementations. This paper presents a methodology for automating the transformation of Interaction Flow Modeling Language (IFML) models into Symfony models using the Atlas Transformation Language (ATL). The proposed approach supports model-driven architecture (MDA) principles and bridges the gap between abstract user interaction models in platform independent model (PIM) level and platform specific model (PSM) level. By leveraging IFML's user-centered modeling capabilities and Symfony's model-view-controller (MVC) framework, our solution enables the automatic generation of reliable, structured front-end architectures. Two case studies demonstrate the feasibility of the approach. This work contributes to the automation of model driven web development by offering a scalable and reusable transformation process from IFML to Symfony.

Corresponding Author:

Abir Sajji
RSI Laboratory, Higher School of Technology, Ibn Tofail University
Kenitra, Morocco
Email: abir.sajji@uit.ac.ma

1. INTRODUCTION

In recent years, model-driven architecture (MDA) has emerged as a powerful paradigm for automating software development through model transformations [1], [2]. A central step in this process is the transition from platform independent model (PIM), which abstract technical details to enhance understanding and flexibility, to platform specific model (PSM) that incorporate the execution and performance details required for deployment [3], [4]. Although many studies have investigated PIM to PSM transformations using UML as the PIM standard, modeling user interactions particularly in front-end or model-view-controller (MVC-based)-based applications remains challenging. The Interaction Flow Modeling Language (IFML) was introduced to address this gap, offering a more intuitive and comprehensive way to represent user interaction flows.

However, the application of IFML to modern PHP frameworks such as Symfony has been rarely explored in the literature. Most existing studies focus on technologies like MongoDB, Laravel, or plain PHP, without fully leveraging the potential of IFML for user-centric modeling. In this work, we selected the Symfony framework to streamline the development of web applications [5]. Symfony is a powerful, secure, and continuously evolving framework whose clear design, intuitive programming approach, and high readability make it accessible to developers regardless of project size or complexity. Unlike Laravel, which emphasizes rapid development through a convention over configuration philosophy, Symfony provides a modular architecture with reusable components, making it particularly well suited for large-scale and complex enterprise applications. Its long-term support releases, extensive community, and adherence to industry standards further reinforce its position as a preferred choice in industrial projects [6], [7].

In our previous work [8], [9], we explored model transformations from the computation independent model (CIM) level represented by BPMN to the PIM level represented by IFML, focusing on generating graphical user interfaces (GUI). Specifically, we utilized the WebRatio platform to automate this process up to the PSM level. However, challenges persist, particularly in customizing interfaces for more complex applications and optimizing the generated systems.

MDA is a concept introduced by the OMG that emphasizes basing software development on standardized models [1]. These models allow developers to focus on the logical design of a program. From these models, it is possible to perform transformations that generate code or produce other models tailored to specific technologies or higher levels of abstraction. MDA provides a structured framework for creating and working with these models.

Models are instances of metamodels, which define a set of concepts and their relationships through class diagrams. The structure of a metamodel is, in turn, represented by a meta metamodel, which is specified using the MOF language [10]. MDA is based on three levels of modeling: the CIM, PIM, and PSM. Model transformation rules are used to move from one modeling level to another, for example, by generating a target model from a source model.

OMG proposed the IFML to specify interaction flow models, which describe how users interact with applications on the front end. The developers of IFML brought over a decade of experience with the Webratio tool and WebML to its creation. In March 2013, OMG officially adopted IFML as a standard [11]. IFML introduces numerous advantages for designing UI across desktop, mobile, and web based applications. Among the five artifacts provided by the IFML specification is the IFML visual syntax [12]. This syntax is simplified and widely recognized by developers, enabling the creation of various tools for generating IFML diagrams.

Symfony [13] is an open source PHP framework originally designed to facilitate the rapid development of robust, modular, and maintainable web applications. Launched in 2005 by SensioLabs, it quickly established itself as one of the leading PHP frameworks, thanks to its component based architecture and philosophy centered on reusability, testability, and flexibility. Symfony promotes an architecture based on the MVC pattern, which separates business logic, presentation logic, and request routing logic. This separation enhances code readability, facilitates collaboration among developers. In addition, Symfony's best practice guides and official documentation provide valuable recommendations for maintaining consistent code organization, implementing effective unit testing, enabling continuous integration, and automating deployment processes.

Despite the growing adoption of MDA for web application development, few studies have focused on fully automating the transformation from IFML to modern PHP frameworks such as Symfony. Existing approaches often suffer from key limitations:

- Limited front-end coverage most focus on back-end or database generation, leaving UI modeling partially addressed.
- Lack of automation many transformations require significant manual intervention, reducing productivity and consistency.

Furthermore, most previous works rely on UML for PIM representation, which is less suited to user interaction modeling compared to IFML. Unlike UML, a general purpose modeling language, IFML is specifically designed to model UI and interaction flows, making it better suited for UI-centric transformations. It provides platform-independent representations of front-end behavior and is supported by dedicated tools, such as WebRatio, that can even generate executable code. This specialization enables more detailed design and greater automation in the development of web and mobile applications [6].

In this context, our study presents a methodology for automating the transformation of IFML models (PIM) into Symfony models (PSM) using the Atlas Transformation Language (ATL) transformation language. ATL was chosen over alternatives such as query/view/transformation (QVT) due to its mature ecosystem, seamless integration with the eclipse modeling framework (EMF), and extensive adoption in the model driven engineering (MDE) community. Its declarative style and native support for OCL facilitate concise and expressive mapping rules, while its performance is adequate for medium-to-large models. Although ATL has a steeper learning curve and is less adapted for direct model-to-code generation compared to Acceleo, its maintainability, portability, and reusability for model to model (M2M) transformations make it a strong fit for this work [14]. Our approach introduces a structured and automated framework that bridges the gap between abstract user interaction models and platform specific implementations. The methodology begins with defining the metamodels for the IFML standard and the Symfony framework into the Eclipse tool. The next step, referred to as M2M transformation, establishes traceability links between the elements of the source and target metamodels. These links, known as transformation rules, are implemented using the ATL language.

The output of this process is a PSM represented in the Ecore format [10]. This generated PSM model serves as the input for the model to code (M2C) transformation, which produces the web application's code. To validate this approach, we conducted two case studies. The final transformation step, M2C, will be addressed in our future work to ensure that the current study remains clear, focused, and easy to follow.

The key contributions of this work can be summarized as follows:

- The implementation of ATL transformation rules to automate the transition from PIM to PSM.
- The validation of our methodology through two case studies on a user profile system and task management system.

The rest of this paper is structured as follows: section 2 reviews related works and discusses existing methodologies of model transformations from PIM to PSM. Section 3 introduces our proposed methodology, including the IFML and Symfony metamodels. Section 4 validates the approach through two case studies. Section 5 compares our approach with other works, and section 6 ends the paper by proposing avenues for future work.

2. RELATED WORKS

This section will discuss the research works that have focused on the transformations from PIM to PSM. In the study, Ouchra *et al.* [15] proposed an approach that introduces an integrated framework based on MDE principles, leveraging ATL to transform a generic metamodel into one tailored for urban satellite image classification. This automated and standardized process ensures a smooth transition from PIM to PSM while enhancing reliability, reproducibility, and efficiency. By structuring satellite data for targeted urban analyses, the method enables the classification of zones such as built-up, cultivated, arid, and water areas, providing a robust tool for urban planning.

Samanipour *et al.* [16] introduce a comprehensive MDA based approach for developing Web 3.0 DApps, extending beyond smart contract development to cover on-chain, off-chain, and communication patterns. Using a land leasing DApp, the method demonstrated automated transformations from BPMN models to code generation on the Ethereum stack. Validation with Epsilon languages and quality metric analysis showed improved ontology coverage, reusability, and comprehensibility compared to prior models. Natek *et al.* [17] defines metamodels for UML and MongoDB, utilizing QVTo scripts to transform UML class diagrams into MongoDB schemas. Testing confirmed the accuracy and integrity of the generated collections, demonstrating the MDA approach's effectiveness in aligning UML based designs with NoSQL databases for robust data management in complex systems.

In their paper Naimi *et al.* [18] present an MDA approach for developing smart tourism web applications, involving three steps: defining a PIM based on the 6 As of tourism, creating a PSM using PHP and clean architecture, and generating source code through XSLT transformations. A case study demonstrates the approach's feasibility in building applications for exploring and booking city attractions, highlighting its benefits. Keskes [19], applies the MDA approach to implement sensitive business processes (SBPs), focusing on identifying and localizing critical knowledge. It transforms SBPs from a conceptual CIM specification to a PIM using an extended BPMN 2.0, integrating knowledge dimensions through core ontologies. PIM to PSM transformations are implemented using MOF 2.0 QVT and mapping rules, enabling precise SBP representation and implementation. Srai and Guerouate [20] combine MDE with NoSQL databases, demonstrating the generation of a PSM model for NoSQL systems. Using a simple class diagram as a case study, it validates the applicability of MDE to NoSQL, leveraging metamodeling and model transformation for effective database design and management.

Meyma *et al.* [21] propose an MDA approach for modernizing e-health applications using QVTo mapping to transform abstract models into concrete ones. The study highlights advancements in eHealth applications by comparing eHealth 2.0 (social web, wikis, blogs) with eHealth 3.0 (AI, semantic web). The development process generates PSM models from PIM models based on UML and IFML standards, producing outputs aligned with ODM and HTML5. Wang and Wang [22], propose a UML-based modeling approach using class and state machine diagrams to analyze the static structure and dynamic behavior of smart contracts in international railway logistics. A conversion algorithm transforms the PIM into a PSM, followed by code generation through a M2C transformation. Experimental results validate the method's feasibility and accuracy. This paper, presented by Mokhtari *et al.* [23] focuses on deriving PIM to PSM in line with the MDA approach. PrivUML, a metamodel for modeling privacy requirements, is transformed into XACML using the QVT transformation language. The process, modifications for success, and an integration proposal for dynamic data anonymization are also discussed.

Erraissi and Banane [24], explore the big data era, where vast amounts of personal and organizational data are generated daily, crucial for decision making. Building on a previously defined generic processing layer metamodel, this paper transforms a big data processing metamodel PIM into a Cloudera-specific metamodel PSM using the ATL transformation language, aligning with MDA architecture. Rahmouni *et al.* [25] proposed an MDA approach for generating Symfony applications from UML PIM models, mainly addressing the back-end structure. Their use of a general-purpose UML metamodel limits native support for modeling user interactions and navigation flows, resulting in partial automation of the front-end layer. In contrast, our approach employs the IFML metamodel, which is specifically designed for UI-centric modeling, enabling direct mappings of constructs such as ViewContainer, ViewComponent, and NavigationFlow to Symfony PSM artifacts through ATL rules. This leads to deeper automation of the transformation process and reduces the need for manual adjustments. Arrhioui *et al.* [26] focus on generating CRUD applications using the CodeIgniter PHP framework. They first model the framework, create a corresponding PIM model, and then use Acceleo templates to generate PHP code for CodeIgniter based CRUD applications.

Ražinskas and Čeponienė [27], present an MDA based methodology for developing Laravel web information systems. It enables developers to generate Laravel code from UML diagrams through PIM to PSM transformations (using a Laravel PSM profile) and PSM to code transformations.

Most related works rely on UML class diagrams at the PIM level, whereas our methodology introduces IFML as a user interaction-oriented standard, making it more intuitive and better suited for front-end and MVC-based applications. Among the reviewed works, Rahmouni *et al.* [25] is the closest to ours; however, their UML-based method primarily targets the back end and provides limited automation for the front-end layer. By contrast, our IFML-based approach enables direct mappings of UI-centric constructs to Symfony PSM artifacts through ATL rules, achieving deeper automation and reducing manual adjustments.

On the PSM side, other approaches target diverse platforms such as MongoDB [17], [20], Laravel [27], while ours focuses on Symfony, a modern and widely adopted PHP framework. This choice ensures seamless alignment with MVC conventions and eliminates the need for ad-hoc mappings. By leveraging ATL, our method achieves a high degree of automation with minimal manual intervention, thereby improving reproducibility, reducing human errors, and enhancing productivity compared to more fragmented or less automated approaches.

3. PROPOSED METHOD

This section describes the proposed approach for generating the Symfony model at the PSM level from the IFML model, by applying transformation rules written in the ATL language while adhering to the MDA approach. Our methodology is structured and pedagogical. The process begins with defining the two metamodels using the Eclipse tool, IFML at the PIM level and Symfony at the PSM level. An IFML model is then created, and transformation rules are applied using the ATL language to generate the Symfony model.

This approach, based on MDA is fully automated, making it highly efficient. It delivers significant productivity gains while maintaining consistency and quality throughout the transformation process. Figure 1 illustrates the MDA transformation process from PIM to PSM. According to its metamodel, the IFML model is transformed into a platform specific Symfony model using the defined ATL rules.

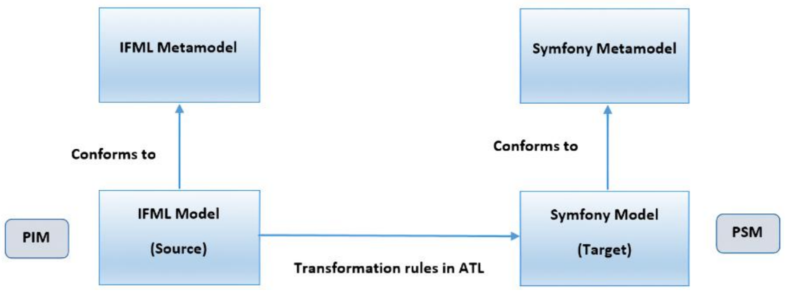
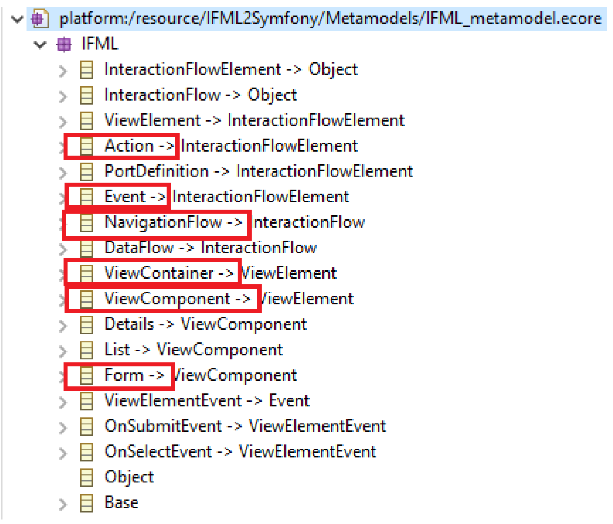


Figure 1. The transformation process from PIM to PSM using ATL

3.1. Interaction Flow Modeling Language metamodel

In this work, we used a simplified IFML metamodel that reduces the complexity of standard IFML elements while preserving the core components necessary for representing UI and their interactions. Advanced or rarely used IFML elements such as *ModuleDefinition*, *Parameter*, *InteractionFlowExpression*, *ThrowingEvent*, *CatchingEvent*, *JumpEvent*, and *SetContextEvent* were omitted, as they were not required by the target Symfony implementation. This simplification reduced the metamodel complexity, improved transformation rule maintainability, and ensured a direct and unambiguous mapping between PIM and PSM elements. The IFML metamodel [28], [29] outlines the structure and behavior of user interactions at an abstract level. Figure 2 illustrates the IFML ecore metamodel, which includes:

- View container: represents high level UI components, such as pages or screens.
- View component: defines the smaller interface elements within containers, like forms or widgets.
- Event: captures interactions triggered by the user, such as button clicks or form submissions.
- Action: represents the logic executed in response to events, such as validating input or saving data.
- Form: represents a UI element specifically designed to collect data or capture user interactions.
- NavigationFlow: specifies the transitions between different parts of the application, ensuring seamless user navigation.



```
platform:/resource/IFML2Symfony/Metamodels/IFML_metamodel.ecore
  IFML
    InteractionFlowElement -> Object
    InteractionFlow -> Object
    ViewElement -> InteractionFlowElement
    Action -> InteractionFlowElement
    PortDefinition -> InteractionFlowElement
    Event -> InteractionFlowElement
    NavigationFlow -> InteractionFlow
    DataFlow -> InteractionFlow
    ViewContainer -> ViewElement
    ViewComponent -> ViewElement
    Details -> ViewComponent
    List -> ViewComponent
    Form -> ViewComponent
    ViewElementEvent -> Event
    OnSubmitEvent -> ViewElementEvent
    OnSelectEvent -> ViewElementEvent
    Object
    Base
```

Figure 2. IFML ecore metamodel

3.2. Symfony framework metamodel

Symfony is a framework designed to simplify web application development by providing a range of features [13], [30]. It organizes web applications by clearly separating display views, server logic, and business rules, ensuring a clean and maintainable structure. Symfony also includes a variety of tools and classes that streamline the creation of complex applications. Moreover, it automates repetitive tasks, allowing developers to concentrate on the unique aspects of their projects. These advantages eliminate the need to build a new web application entirely from scratch each time.

We adopted the Symfony metamodel proposed by Rahmouni *et al.* [25], which comprises a set of artifacts represented by a UML class diagram. Figure 3 illustrates the metamodel of the Symfony framework. The various metaclasses that form the target metamodel of the Symfony framework, along with their concepts, are outlined below:

- Symfony package: represents the concept of a package within the Symfony framework.
- Models: correspond to the collection of models within the MVC2 architectural design.
- Model: represents the "model" class in the context of the MVC2 pattern.
- Controllers: refers to the controller package in the MVC2 architecture.
- Abstract controller: an optional base controller that can be extended to access predefined helper methods.
- Controller: represents the "controller" class as defined in the MVC2 architecture.
- Views: a view package that groups various template classes as part of the MVC2 architectural structure.
- Template: represents the view concept in the MVC2 framework, enabling the separation of HTML and PHP code.
- Actions: reflect the notion of a package containing functions.
- Function: represents the concept of the action class within the PHP Symfony framework.
- Forms: refers to the collection of form classes.
- Form: represents the form class, modeling the concept of a form in the MVC2 architectural pattern.
- Router: facilitates the mapping of a URL (or URL patterns) to a specific controller method, referred to as an action.

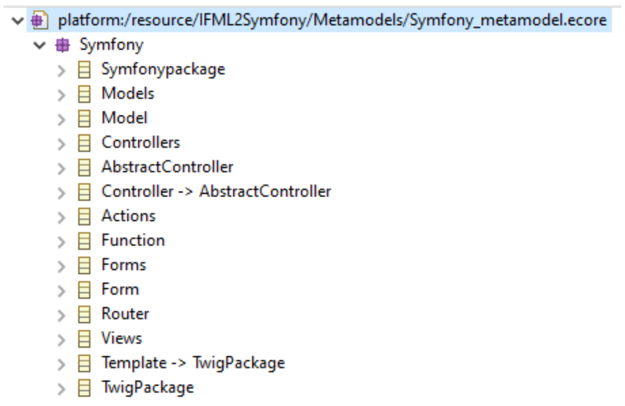

```
platform:/resource/IFML2Symfony/Metamodels/Symfony_metamodel.ecore
  Symfony
    Symfonypackage
    Models
    Model
    Controllers
    AbstractController
    Controller -> AbstractController
    Actions
    Function
    Forms
    Form
    Router
    Views
    Template -> TwigPackage
    TwigPackage
```

Figure 3. Symfony ecore metamodel

3.3. Transformation rules

The target model is generated guided by the specifications of transformation rules. The latter aligns some elements in the source model, and based on these elements, generates corresponding elements in the target model. In this section, we outline the traceability relationships between the metaclasses of the source and target metamodels. Each traceability relationship corresponds to a transformation rule implemented using the ATL transformation language [14].

Model transformation rules manage the mapping between components of the source and target metamodels. This part outlines the traceability relationships between the metaclasses that constitute the IFML and PHP Symfony metamodels. These transformation rules enable the conversion of an IFML model into a PHP Symfony model detailed in Table 1.

Table 1. Description of the transformation rules from IFML to Symfony

IFML source element	Symfony target artifact	Description
ViewContainer	Controller	Represents a logical unit of the UI that groups multiple views or components. In Symfony, mapped to a Controller that handles requests and returns views.
ViewComponent	Template and model	Represents a specific UI element or view within a container. In Symfony, mapped to a Twig template for the view and a model for data handling.
Event	Form	Captures user actions such as clicks or submissions. In Symfony, represented as a form class handling validation and data submission.
Action	Function	Defines backend logic triggered by events. In Symfony, implemented as a function.
NavigationFlow	Router	Defines navigation between views. In Symfony, mapped to routing rules in routes.

After defining the various transformation rules for converting the IFML model into the Symfony model, we will describe these rules in the ATL language. This step involves detailing how the transformation rules are implemented and specifying the mappings between the elements of the IFML metamodel and those of the Symfony metamodel within ATL. Algorithm 1 illustrates the transformation rules in the ATL language.

Algorithm 1. The ATL mapping rules converting the IFML model into the Symfony model

```
rule ViewContainer2Controller {
  from
       vc : IFML!ViewContainer
    to
        c : Symfony!Controller (
             name <- vc.name
        )
}

rule ViewComponent2TemplateAndModel {
  from
       vc : IFML!ViewComponent
    to
        tmp : Symfony!Template (
           name <- vc.name + 'Template.html.twig',
           attribute <- vc.attribute
        ),
          mdl : Symfony!Model (
           name <- vc.name + 'Model'
        )
}

rule NavigationFlow2Router {
    from
        nf : IFML!NavigationFlow
    to
        router : Symfony!Router (
            name <- nf.name + 'Route',
            source <- Symfony!Controller.allInstances()
                      ->select(c | c.name = nf.sourceInteractionFlowElement.name +
'Controller')
                      ->first(),
            target <- Symfony!Controller.allInstances()
                      ->select(c | c.name = nf.targetInteractionflowelement.name +
'Controller')
                      ->first()
        )
}

rule Event2Form {
    from
        evt : IFML!Event
    to
        frm : Symfony!Form (
        name <- evt.name + 'Form'
        )
}

rule Action2Function {
    from
        act : IFML!Action
    to
        funt : Symfony!Function (
        name <- act.name + 'Action'
        )
}
```

The transformation rules in this approach are designed for full automation, but an IT specialist may perform manual validation to ensure structural conformance (correct IFML-Symfony mappings and naming), for example, ambiguities may arise when two ViewComponent elements share the same name, causing naming conflicts in generated controllers, semantic accuracy (intended behavior of navigation flows and event-triggered actions), and completeness (no missing elements). The process involves loading the generated PSM in Eclipse, running EMF/OCL checks, inspecting key artifacts (controllers, forms, and routes), adjusting mappings if needed, and re-running the transformation.

To better illustrate how these transformation rules operate in practice, we provide below a detailed walk-through of two representative rules, namely ViewComponent2TemplateAndModel and NavigationFlow2Router.

ViewComponent2TemplateAndModel: this rule takes an IFML ViewComponent as input (from) and generates two corresponding Symfony artifacts (to): a template and a model. The name attribute of the source component is reused to construct the names of both target elements by appending suffixes (Template.html.twig and Model). Additionally, attributes of the IFML component are directly transferred to the Symfony Template. This ensures that each UI element in IFML is systematically represented by both a visual template and a data model in Symfony.

The NavigationFlow2Router rule transforms an IFML NavigationFlow into a Symfony router. The router’s name is built by appending "Route" to the flow name (name <- nf.name + 'Route'). To set the router’s endpoints, the rule scans all generated Symfony controllers (allInstances()), filters them by name (select) to find the controller whose name equals the flow endpoint name plus "Controller", and takes the first match (first()). Concretely, source is matched against nf.sourceInteractionFlowElement.name + 'Controller', and target against nf.targetInteractionflowelement.name + 'Controller'. This establishes explicit links from the router to the controllers corresponding to the flow’s origin and destination, ensuring correct navigation semantics on the target platform.

4. CASE STUDIES

We present two case studies to demonstrate and validate our approach using two IFML models of a user profile system and task management system at the PIM level. The transformation results in an ecore model, which represents the PSM level.

4.1. The platform independent model level

The user profile management system model focuses on managing user profiles, including features like authentication, profile viewing, and editing. It begins with key elements defined in IFML, such as view containers: the LoginPage, HomePage, ProfilePage, and WelcomePage, which serve as the primary UI. Each container includes View Components, like the LoginForm where users enter their credentials, and the UserProfile, which displays detailed user information. The process also involves events such as SubmitLogin, triggered when users interact with forms, and EditProfile, which allows users to update their profiles.

The model uses forms, like the EditProfileForm, to manage user inputs for updating personal information. It also specifies actions like ValidateLogin, which checks user credentials; LoadProfileForEditing, which retrieves existing profile data; and SaveProfile, which saves any changes made to user information. Navigation flows seamlessly link these components and actions to enhance the user experience.

For example, LoginToHomeFlow redirects users from the login page to the homepage after successful authentication, while ProfileToEditProfileFlow and EditProfileToProfileFlow manage transitions between profile viewing and editing functionalities. Figure 4 shows the IFML model which was defined using the IFML metamodel.

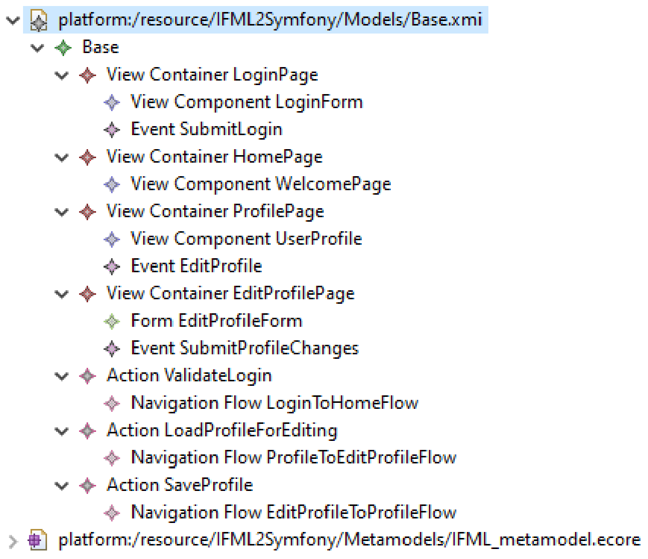
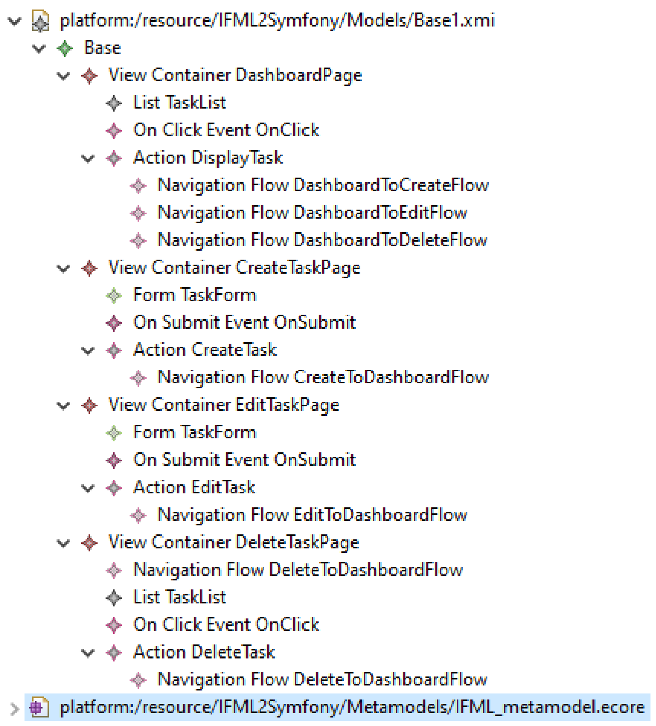

```
platform:/resource/IFML2Symfony/Models/Base.xmi
  Base
    View Container LoginPage
      View Component LoginForm
      Event SubmitLogin
    View Container HomePage
      View Component WelcomePage
    View Container ProfilePage
      View Component UserProfile
      Event EditProfile
    View Container EditProfilePage
      Form EditProfileForm
      Event SubmitProfileChanges
    Action ValidateLogin
      Navigation Flow LoginToHomeFlow
    Action LoadProfileForEditing
      Navigation Flow ProfileToEditProfileFlow
    Action SaveProfile
      Navigation Flow EditProfileToProfileFlow
platform:/resource/IFML2Symfony/Metamodels/IFML_metamodel.ecore
```

Figure 4. IFML model of user profile management system

The task management system enables users to create, view, edit, and delete tasks. It consists of four main view containers: DashboardPage (displays all tasks), CreateTaskPage (task creation), EditTaskPage (task modification), and DeleteTaskPage (task deletion). Each container includes view components such as TaskList, which lists tasks, and TaskForm, which captures task details (title, date, and description). User-triggered events include OnSubmit and OnClick, linked to actions like DisplayTask, CreateTask, EditTask, and DeleteTask.

Navigation flows ensure smooth transitions: from dashboard to creation (DashboardToCreateFlow), creation to dashboard (CreateToDashboardFlow), from dashboard to editing (DashboardToEditFlow), and returning after editing (EditToDashboardFlow), from dashboard to deleting (DashboardToDeleteFlow), and returning after deleting (DeleteToDashboardFlow). Figure 5 shows the IFML model representing this task management flow.



```
platform:/resource/IFML2Symfony/Models/Base1.xmi
  Base
    View Container DashboardPage
      List TaskList
      On Click Event OnClick
      Action DisplayTask
        Navigation Flow DashboardToCreateFlow
        Navigation Flow DashboardToEditFlow
        Navigation Flow DashboardToDeleteFlow
    View Container CreateTaskPage
      Form TaskForm
      On Submit Event OnSubmit
      Action CreateTask
        Navigation Flow CreateToDashboardFlow
    View Container EditTaskPage
      Form TaskForm
      On Submit Event OnSubmit
      Action EditTask
        Navigation Flow EditToDashboardFlow
    View Container DeleteTaskPage
      Navigation Flow DeleteToDashboardFlow
      List TaskList
      On Click Event OnClick
      Action DeleteTask
        Navigation Flow DeleteToDashboardFlow
platform:/resource/IFML2Symfony/Metamodels/IFML_metamodel.ecore
```

Figure 5. IFML model of task management system

4.2. The platform specific model level

The Symfony models generated at the PSM level, as shown in Figure 6 (see in Appendix), are the result of transforming the IFML models into Symfony models using transformation rules written in the ATL language. Those models adhere to Symfony's architectural principles and translate the abstract elements defined in IFML into practical artifacts tailored for web application development in Symfony. The main components of the PSM model include:

- Controllers: responsible for managing the application logic.
- Models: manage the data structure and business logic.
- Templates: define the presentation layer, determining how data is displayed.
- Forms, functions, and routers: facilitate seamless user interaction and navigation within the application.

To make the traceability explicit, Figure 6 juxtaposes the IFML models and the generated Symfony PSM, annotating corresponding elements with the same color/number. Controllers (1) are produced from IFML ViewContainers by the ViewContainer2Controller rule; templates and models (2) are produced from ViewComponents by ViewComponent2TemplateAndModel-note that List and Form are subclasses of ViewComponent in our metamodel, hence they also yield a template and a model. Forms (3) are produced from Events by Event2Form; Functions (4) from actions by Action2Function; and Routers (5) from NavigationFlows by NavigationFlow2Router, which resolves the route’s source/target to the corresponding controllers. For example, Controller LoginPage is generated from the IFML ViewContainer LoginPage, and LoginFormTemplate.html.twig with LoginFormModel are generated from the ViewComponent LoginForm; this visual linkage demonstrates how each PSM artifact directly derives from its IFML element according to the rules in Algorithm 1.

This transformation demonstrates the effectiveness of the MDA approach in automating the generation of platform specific software components. It ensures consistency, scalability, and maintainability, making it an invaluable framework for web application development.

5. DISCUSSION

This section compares our approach with the state of the art. Our methodology aims at the automation of the transformation from IFML PIM to Symfony PSM, leveraging the principles of MDA. This is particularly significant because it addresses the lack of standardized approaches in the literature.

It provides, in particular, a systematic approach for the definition of traceability links between source and target metamodels in compliance with the MDA approach, ensuring both clarity and reproducibility. Another positive aspect is that our methodology applies ATL rules in the automation process to facilitate smooth automation, reduce human errors, and thus enhance productivity.

While existing PIM to PSM approaches differ in modeling languages, target platforms, and transformation languages, most rely on UML as the PIM standard. UML, being general purpose, lacks dedicated constructs for user interaction flows, leading to partial coverage of UI aspects or reliance on ad-hoc mappings. IFML, by contrast, is explicitly designed for modeling navigation, events, and view compositions, enabling direct and systematic mappings to platform specific front-end artifacts. Rahmouni *et al.* [25], for instance, also target Symfony but primarily address back-end generation, with limited automation for the UI. Our methodology leverages IFML and ATL to automate both UI and back-end elements in Symfony, reducing manual intervention and improving consistency, as detailed in Table 2.

Table 2. Comparison of PIM to PSM approaches

Paper	PIM level	PSM level	Transformation language
Ouchra *et al.* [15]	UML class diagram	UML class diagram	ATL
Natek *et al.* [17]	UML class diagram	MongoDB	QVT
Naimi *et al.* [18]	UML class diagram	PHP	ATL
Srai and Guerouate [20]	UML class diagram	MongoDB	QVT
Meyma *et al.* [21]	UML and IFML	ODM and HTML5	QVT
Mokhtari *et al.* [23]	PrivUML	XACML	QVT
Erraissi and Banane [24]	big data	Cloudera	ATL
Rahmouni *et al.* [25]	UML class diagram	Symfony	ATL
Ražinskas and Čeponienė [27]	UML class diagram	Laravel	ATL
Our methodology	IFML	Symfony	ATL

As shown in Table 2, our methodology stands out by combining IFML's UI-focused modeling with ATL based automation for Symfony. This ensures both front-end and back-end coverage, with structured and reproducible transformations that address gaps in prior works, particularly those limited to UML-based or back-end-centric approaches. This methodology addresses the lack of automated tools for IFML to Symfony transformations and lays a foundation for further advancements in MDA applications by enabling structured and reproducible transformations.

6. CONCLUSION

In this work, we presented a methodology to automate the transformation of IFML models PIM into Symfony models PSM using the ATL transformation language. Our work fills a significant gap in the existing literature by introducing IFML as a more intuitive standard for representing user interactions, particularly in front-end and MVC-based applications. By defining the Symfony metamodel and creating clear transformation rules, we showed how PSM can be generated efficiently and systematically.

We validated our approach through two case studies on a user profile system and task management system, demonstrating the practicality and reliability of our methodology. ATL enabled an automated transformation process, reducing manual effort and ensuring greater consistency in the generated models.

However, the approach still has limitations: it relies entirely on ATL, only partially supports advanced IFML constructs such as conditional navigation flows, and lacks runtime validation of generated applications. As future work, we plan to extend the methodology beyond the PIM to the PSM phase by incorporating executable code generation M2C using Acceleo, enabling full automation of Symfony web application development.

We also aim to expand rule coverage to advanced IFML elements and introduce runtime validation to ensure behavioral correctness. Furthermore, future work will include a quantitative evaluation of the methodology's effectiveness, measuring metrics such as transformation time, number of elements correctly transformed, degree of automation, and manual effort saved compared to baseline manual mapping. This will provide a more rigorous scientific assessment of scalability and reliability.

To conclude, our methodology makes a valuable contribution to MDA by bridging IFML and Symfony. It provides a structured and automated solution that simplifies and accelerates the development of modern web applications.

FUNDING INFORMATION

The authors express their gratitude to the Scientific Production Support Center of Ibn Tofail University for its financial support of this research. This funding has contributed to covering publication costs and enhancing the dissemination of the study's findings.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Abir Sajji	✓	✓	✓		✓	✓		✓	✓				✓	✓
Yassine Rhazali	✓	✓		✓		✓	✓	✓		✓	✓	✓	✓	✓
Youssef Hadi	✓	✓		✓			✓	✓		✓	✓	✓	✓	✓

C	: **C**onceptualization	I	: **I**nvestigation	Vi	: **Vi**sualization
M	: **M**ethodology	R	: **R**esources	Su	: **Su**pervision
So	: **So**ftware	D	: **D**ata Curation	P	: **P**roject administration
Va	: **Va**lidation	O	: Writing - **O**riginal Draft	Fu	: **Fu**nding acquisition
Fo	: **Fo**rmal analysis	E	: Writing - Review & **E**diting		

CONFLICT OF INTEREST STATEMENT

The authors state no conflict of interest.

DATA AVAILABILITY

Data availability is not applicable to this paper as no new data were created or analyzed in this study.

REFERENCES

[1] OMG.Org, "Model Driven Architecture (MDA) | Object Management Group," *Https://www.Omg.Org/Mda/*, 2019. https://www.omg.org/mda/. (Date accessed Jan. 01, 2025).

[2] X. Blanc, *MDA in Action*: *Model-Driven Software Engineering*. Paris, France: Hermes Science Publications, 2005.

[3] N. Silega, M. Noguera, Y. I. Rogozov, V. S. Lapshin, and T. Gonzalez, "Transformation From CIM to PIM: a Systematic Mapping," *IEEE Access*, vol. 10, pp. 90857–90872, 2022, doi: 10.1109/ACCESS.2022.3201556.

[4] D. Gall, "Efficiency Requirements in PIM to PSM Transformations," in *Proceedings of the 2nd International Workshop on Model-Driven Architecture and Modeling Theory-Driven Development*, 2010, pp. 93–98, doi: 10.5220/0003044800930098.

[5] S. Sherin, A. Muqeet, M. U. Khan, and M. Z. Iqbal, "QExplore: An exploration strategy for dynamic web applications using guided search," *Journal of Systems and Software*, vol. 195, p. 111512, Jan. 2023, doi: 10.1016/j.jss.2022.111512.

[6] M. Laaziri, K. Benmoussa, S. Khoulji, K. M. Larbi, and A. El Yamami, "A comparative study of Laravel and Symfony PHP frameworks," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 1, pp. 704-712, Feb. 2019, doi: 10.11591/ijece.v9i1.pp704-712.

[7] H. Abutaleb, A. Tamimi, and T. Alrawashdeh, "Empirical Study of Most Popular PHP Framework," in *2021 International Conference on Information Technology (ICIT)*, Jul. 2021, pp. 608–611, doi: 10.1109/ICIT52682.2021.9491679.

[8] A. Sajji, Y. Rhazali, and Y. Hadi, "IFML-based graphical user interfaces generated from BPMN up to PSM level," in *Enhancing Performance, Efficiency, and Security Through Complex Systems Control*, 2024, pp. 201–222, doi: 10.4018/979-8-3693-0497-6.ch012.

[9] A. Sajji, Y. Rhazali, and Y. Hadi, "Graphical user Interfaces Generation from BPMN (Business Process Model and Notation) via IFML (Interaction Flow Modeling Language) up to PSM (Platform Specific Model) Level," *International Journal of Advanced Computer Science and Applications*, vol. 14, no. 2, pp. 419–431, 2023, doi: 10.14569/IJACSA.2023.0140251.

[10] Object Management Group (OMG), "About the Meta Object Facility Specification Version 2.0," 2006. [Online]. Available: https://www.omg.org/spec/MOF/2.0/. (Date accessed: Sep. 10, 2025).

[11] C. Bernaschina, M. Brambilla, A. Mauri, and E. Umuhoza, "A big data analysis framework for model-based web user behavior analytics," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer, Cham, 2017, vol. 10360, pp. 98–114, doi: 10.1007/978-3-319-60131-1_6.

[12] R. Acerbis, A. Bongio, M. Brambilla, and S. Butti, "Model-driven development based on OMG's IFML with WebRatio Web and mobile platform," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer International Publishing, Cham, vol. 9114, pp. 605–608, 2015, doi: 10.1007/978-3-319-19890-3_39.

[13] Symfony SAS, "Symfony Documentation," 2023. https://symfony.com/doc/current/index.html. (Date accessed Jan. 01, 2025).

[14] The Eclipse Foundation, "ATL - The Eclipse Foundation," *The Eclipse Foundation*, 2012. https://eclipse.dev/atl/. (Date accessed Jan. 01, 2025).

[15] H. Ouchra, A. Belangour, A. Erraissi, and M. Labied, "New MDA Transformation Process from Urban Satellite Image Classification to Specific Urban Landsat Satellite Image Classification," *Journal of Environmental & Earth Sciences*, vol. 7, no. 1, pp. 81–91, Oct. 2024, doi: 10.30564/jees.v7i1.7145.

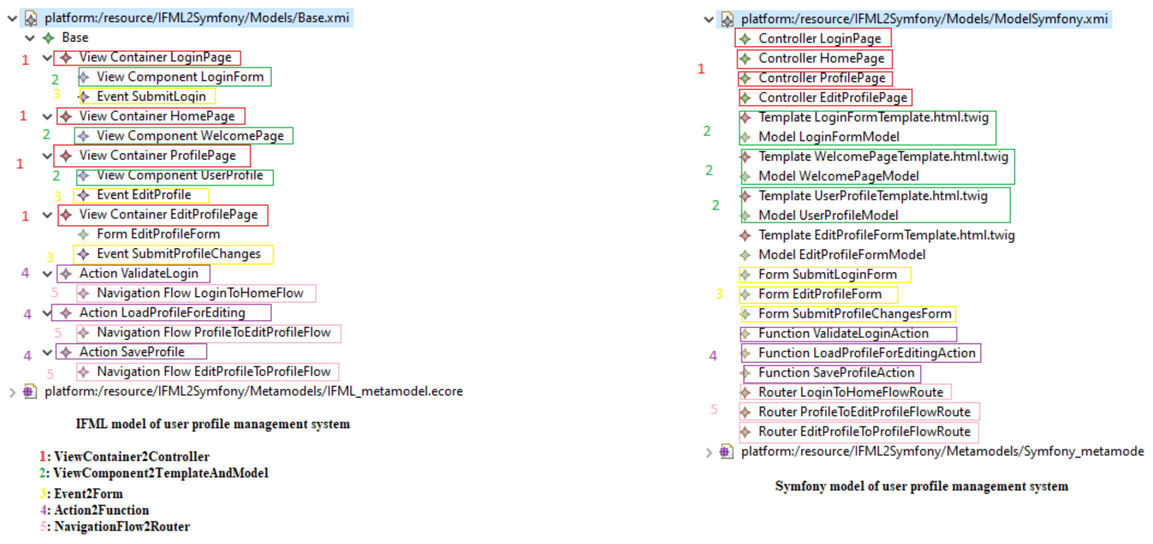
[16] A. Samanipour, O. Bushehrian, and G. Robles, "MDAPW3: MDA-based development of blockchain-enabled decentralized applications," *Science of Computer Programming*, vol. 239, p. 103185, Jan. 2025, doi: 10.1016/j.scico.2024.103185.

[17] H. Natek, A. Srai, A. Badaoui, and F. Gueroaute, "A Model-Driven Approach To Transform Uml Models Into Mongodb Schemas Using Qvto: From Pim To Psm," *Journal of Theoretical and Applied Information Technology*, vol. 102, no. 15, pp. 5928–5938, 2024.

[18] L. Naimi, E. M. Bouziane, and A. Jakimi, "A Model-Driven Approach for Developing Smart Tourism Web Applications," in

Lecture Notes in Networks and Systems, vol. 1100, 2024, pp. 255–263, doi: 10.1007/978-3-031-68660-3_24.
[19] M. Keskes, "A Model Driven Architecture Approach for Implementing Sensitive Business Processes," in *Lecture Notes in Business Information Processing*, vol. 486, pp. 227–242, 2024, doi: 10.1007/978-3-031-51664-1_16.
[20] A. Srai and F. Guerouate, "Towards the Generation of a PSM Model from a PIM Model, Integration of the MDA Approach in NoSQL Databases, the Case of Document-Oriented NoSQL Platforms," *International Journal of Engineering Trends and Technology*, vol. 71, no. 5, pp. 146–155, May 2023, doi: 10.14445/22315381/IJETT-V71I5P215.
[21] M. M. Meyma, N. Laaz, and S. Mbarki, "Upgrading eHealth 2.0 Applications to eHealth 3.0 Based on QVTo Mapping," in *Proceedings - 2023 International Conference on Digital Age and Technological Advances for Sustainable Development, ICDATA 2023*, May 2023, pp. 28–35, doi: 10.1109/ICDATA58816.2023.00015.
[22] Z. Wang and K. Wang, "Design of international railway logistics management system based on MDA and smart contract," in *Third International Conference on Green Communication, Network, and Internet of Things (CNIoT 2023)*, Oct. 2023, p. 95, doi: 10.1117/12.3011076.
[23] J. El Mokhtari, A. A. El Kalam, S. Benhaddou, and J. P. Leroy, "Transformation of PrivUML into XACML Using QVT," in *Advances in Intelligent Systems and Computing*, vol. 1383, pp. 984–996, 2021, doi: 10.1007/978-3-030-73689-7_93.
[24] A. Erraissi and M. Banane, "Managing Big Data using Model Driven Engineering: From Big Data Meta-model to Cloudera PSM meta-model," in *2020 International Conference on Decision Aid Sciences and Application, DASA 2020*, Nov. 2020, pp. 1235–1239, doi: 10.1109/DASA51403.2020.9317292.
[25] M. Rahmouni, C. Talbi, and S. Ziti, "Model-driven architecture: generating models from Symfony framework," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 30, no. 3, pp. 1659-1668, Jun. 2023, doi: 10.11591/ijeecs.v30.i3.pp1659-1668.
[26] K. Arrhioui, S. Mbarki, O. Betari, S. Roubi, and M. Erramdani, "A Model Driven Approach for Modeling and Generating PHP CodeIgniter based Applications," *Transactions on Machine Learning and Artificial Intelligence*, vol. 5, no. 4, 2017, doi: 10.14738/tmlai.54.3189.
[27] M. Ražinskas and L. Čeponiene, "MDA approach for laravel framework code generation from UML diagrams," *CEUR Workshop Proceedings*, vol. 2698, 2020.
[28] M. Brambilla, E. Umuhoza, and R. Acerbis, "Model-driven development of user interfaces for IoT systems via domain-specific components and patterns," *Journal of Internet Services and Applications*, vol. 8, no. 1, pp. 1-21, Dec. 2017, doi: 10.1186/s13174-017-0064-1.
[29] O. M. Group, "About the Interaction Flow Modeling Language Specification Version 1.0," 2015. [Online]. Available: http://www.omg.org/spec/IFML/1.0/. (Date accessed: Jan. 01, 2025).
[30] F. Potencier, *Symfony 5: The Fast Track*. Paris, France: SensioLabs, 2020, ISBN: 9782918390374.

APPENDIX



```
platform:/resource/IFML2Symfony/Models/Base.xmi
  Base
1   View Container LoginPage
2     View Component LoginForm
3     Event SubmitLogin
1   View Container HomePage
2     View Component WelcomePage
1   View Container ProfilePage
2     View Component UserProfile
3     Event EditProfile
1   View Container EditProfilePage
      Form EditProfileForm
3     Event SubmitProfileChanges
4   Action ValidateLogin
5     Navigation Flow LoginToHomeFlow
4   Action LoadProfileForEditing
5     Navigation Flow ProfileToEditProfileFlow
4   Action SaveProfile
5     Navigation Flow EditProfileToProfileFlow
platform:/resource/IFML2Symfony/Metamodels/IFML_metamodel.ecore
```

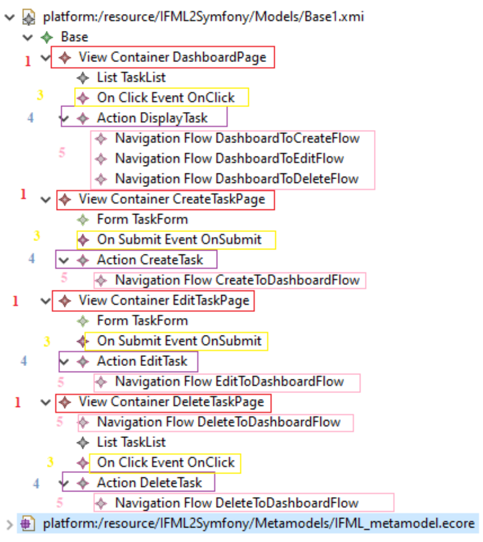
IFML model of user profile management system

1: ViewContainer2Controller
2: ViewComponent2TemplateAndModel
3: Event2Form
4: Action2Function
5: NavigationFlow2Router

```
platform:/resource/IFML2Symfony/Models/ModelSymfony.xmi
    Controller LoginPage
    Controller HomePage
1   Controller ProfilePage
    Controller EditProfilePage
2   Template LoginFormTemplate.html.twig
    Model LoginFormModel
2   Template WelcomePageTemplate.html.twig
    Model WelcomePageModel
2   Template UserProfileTemplate.html.twig
    Model UserProfileModel
    Template EditProfileFormTemplate.html.twig
    Model EditProfileFormModel
    Form SubmitLoginForm
3   Form EditProfileForm
    Form SubmitProfileChangesForm
    Function ValidateLoginAction
4   Function LoadProfileForEditingAction
    Function SaveProfileAction
    Router LoginToHomeFlowRoute
5   Router ProfileToEditProfileFlowRoute
    Router EditProfileToProfileFlowRoute
platform:/resource/IFML2Symfony/Metamodels/Symfony_metamode
```

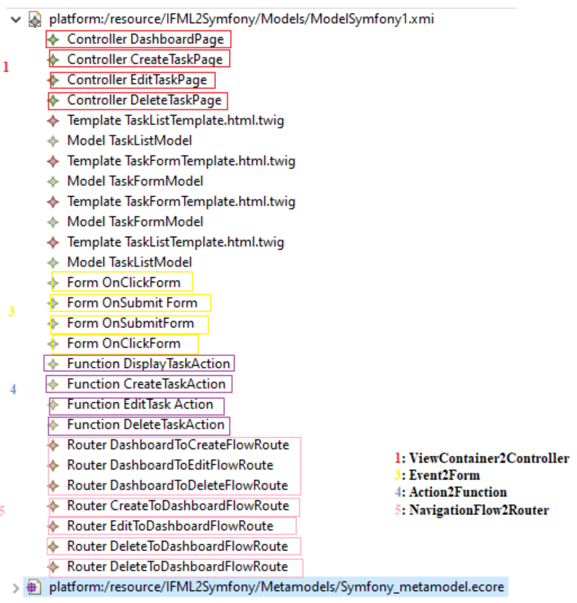
Symfony model of user profile management system

Figure 6. Symfony models for the user profile and task management systems



```
platform:/resource/IFML2Symfony/Models/Base1.xmi
  Base
1   View Container DashboardPage
      List TaskList
3     On Click Event OnClick
4     Action DisplayTask
5       Navigation Flow DashboardToCreateFlow
        Navigation Flow DashboardToEditFlow
        Navigation Flow DashboardToDeleteFlow
1   View Container CreateTaskPage
      Form TaskForm
3     On Submit Event OnSubmit
4     Action CreateTask
5       Navigation Flow CreateToDashboardFlow
1   View Container EditTaskPage
      Form TaskForm
3     On Submit Event OnSubmit
4     Action EditTask
5       Navigation Flow EditToDashboardFlow
1   View Container DeleteTaskPage
5     Navigation Flow DeleteToDashboardFlow
      List TaskList
3     On Click Event OnClick
4     Action DeleteTask
5       Navigation Flow DeleteToDashboardFlow
platform:/resource/IFML2Symfony/Metamodels/IFML_metamodel.ecore
```

IFML model of task management system



```
platform:/resource/IFML2Symfony/Models/ModelSymfony1.xmi
    Controller DashboardPage
1   Controller CreateTaskPage
    Controller EditTaskPage
    Controller DeleteTaskPage
    Template TaskListTemplate.html.twig
    Model TaskListModel
    Template TaskFormTemplate.html.twig
    Model TaskFormModel
    Template TaskFormTemplate.html.twig
    Model TaskFormModel
    Template TaskListTemplate.html.twig
    Model TaskListModel
    Form OnClickForm
3   Form OnSubmit Form
    Form OnSubmitForm
    Form OnClickForm
    Function DisplayTaskAction
4   Function CreateTaskAction
    Function EditTask Action
    Function DeleteTaskAction
    Router DashboardToCreateFlowRoute
    Router DashboardToEditFlowRoute
    Router DashboardToDeleteFlowRoute
5   Router CreateToDashboardFlowRoute
    Router EditToDashboardFlowRoute
    Router DeleteToDashboardFlowRoute
    Router DeleteToDashboardFlowRoute
platform:/resource/IFML2Symfony/Metamodels/Symfony_metamodel.ecore
```

Symfony model of task management system

Figure 6. Symfony models for the user profile and task management systems *(continued)*

BIOGRAPHIES OF AUTHORS

Abir Sajji Ibn Tofail University awarded her a Master's Degree in Software Quality at the Faculty of Sciences, Kenitra, Morocco, in 2010. She then worked at the presidency of the same university as Head of the Student Affairs and Academic Services Department at the Presidency. She is also pursuing her Ph.D. degree at Ibn Tofail University, Faculty of Sciences, Kenitra, Morocco, since 2021. Her main research subjects are software processes and conceptual modeling. She can be contacted at email: abir.sajji@uit.ac.ma.

Yassine Rhazali is a Moroccan professor. He received his Ph.D. in Computer Science from Ibn Tofail University, Kenitra, Morocco, more precisely in Software Engineering. His specialty is model driven engineering; he proposed several approaches in model engineering. His approaches have been validated in various scientific papers published at international scientific conferences and international scientific journals. Currently, he is a professor-researcher, reviewer, editorial board member, and scientific committee member in several international scientific journals and at various international scientific conferences. He taught computer science at different levels in several universities and institutes. He can be contacted at email: dr.yassine.rhazali@gmail.com.

Youssef Hadi did his Ph.D. in 2008 at Mohammed V University, Faculty of Sciences, Rabat, Morocco. Presently, he works as Assistant Director for Educational Affairs at EST Kenitra, Morocco, and heads many sections at Master's Formations. He heads Software Engineering and Multimedia Database Indexing research teams. In such areas of research, he has made significant contributions and publications. He can be contacted at email: hadi@uit.ac.ma.