Journal of Software Engineering and Applications, 2014, 7, 360-370
Published Online May 2014 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2014.75033

Scientific Research

Assessing a Model-Driven Web-Application Engineering Approach

Ali Fatolahi, Stéphane S. Somé

School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada
Email: ssome@eecs.uottawa.ca

Received 18 March 2014; revised 15 April 2014; accepted 23 April 2014

Abstract

Model-Driven Engineering (MDE) by reframing software development as the transformation of high-level models, promises lots of gains to Software Engineering in terms of productivity, quality and reusability. Although a number of empirical studies have established the reality of these gains, there are still lots of reluctances toward the adoption of MDE in practice. This resistance can be explained by several technological and social factors among which a natural scepticism toward novel approaches. In this paper we attempt to provide arguments to help alleviate this scepticism by conducting an assessment of a MDE approach. Our goal is to show that although this MDE is novel, it retains similarities with the conventional Software Engineering approach while automating aspects of it.

Keywords

Model-Driven Engineering (MDE), Software Process Assessment, Web-Engineering

1. Introduction

Model-Driven Engineering (MDE) is a paradigm for software development at the core of which abstract models are used to describe software and systematically transformed to more concrete models up to executable code. MDE raises the abstraction level of languages needed to develop software. It shields software developers from the complexities of underlying implementation platforms [1]. MDE approaches offer several potential benefits to Software Engineering including improved productivity, portability, maintainability and interoperability [2]. However, there are still some resistances in organizations for the wide adoption of MDE as illustrated by surveys such as [3] in which only about 13% of respondents reported that they always use modelling, or [4] in which 35 out of 50 professional software engineers in 50 companies reported no use of the Unified Modelling Language (UML). The adoption of MDE is hindered by several factors including social and technical issues [5].

How to cite this paper: Fatolahi, A. and Somé, S.S. (2014) Assessing a Model-Driven Web-Application Engineering Approach. *Journal of Software Engineering and Applications*, **7**, 360-370. http://dx.doi.org/10.4236/jsea.2014.75033

As put by France and Rumpe [1], the realization of the MDE vision is a wicked problem that requires tackling a variety of interrelated social and technical problems.

The slow rate of adoption of MDE is partly due to the scepticism of software developers and managers toward new approaches. Practitioners tend to adapt to their favourite approaches and resist to new "revolutionary" ways of developing software [6]. It is therefore important that newly proposed approaches provide arguments of efficiency and non-disturbance in order to improve their chance of adoption.

We developed a Model-Driven Engineering approach for Web-applications called MODEWIS (Model-Driven Development of Web Information Systems) [7]. This MDE approach adopts the OMG's Model Driven Architecture (MDA) [8] principles where a distinction is made between three conventional levels of abstraction: the Computation Independent Model (CIM) level, the Platform Independent Model (PIM) level, and the Platform Specific Model (PSM) level. We further distinguish two levels within the PSM: an Abstract-Platform Specific Model (APSM) [9] level and a Specific-Platform Specific Model (SPSM). The APSM is concerned with models specified with respect to common features of different web platforms with no reference to specific implementation platforms, while the Specific-Platform Specific Model (SPSM) integrates these implementation details. The distinction between APSM and SPSM allows the reuse of generic transformations PIM-to-APSM regardless of the implementation platform as well as model portability. Our objective in this paper is to present an assessment of MODEWIS in regard to 1) closeness to "classical" software engineering (familiarity), 2) capacity to "mimic" human design decisions (correctness of design inferences) and 3) gain in productivity. We believe that showing that a MDE remains close to traditional software engineering—known to practitioners, and that the transformations embedded are not a complete departure from the manual design decisions in those approaches could alleviate the natural scepticism of practitioners. Showing that there is a potential gain in productivity would provide additional clues to the benefits of the approach.

The rest of this paper is organized as follows. Section 2 contains a discussion on the related work. In Section 3, we present a summary of MODEWIS. In Section 4, we compare MODEWIS to the conventional Software Engineering activities. The design inference capabilities of MODEWIS are assessed in Section 5, and the productivity of the approach in terms of automatically created model elements versus provided elements, is discussed in Section 6. Finally, Section 7 concludes this paper.

2. Related Work

Different studies have been conducted on the evaluation of Model-Driven Engineering. Most of these have examined empirically the degree of adoption of MDE in industries, as well as attempted to identify the advantages and limitations of MDE in practice. In [10], Mohagheghi and Dehlen present a review of 25 papers on such empirical studies published between 2000 and 2007. This work has found that the practice of MDE is mainly focused on code generation but did not find enough evidence about the productivity benefit of MDE for large projects. Mohagheghi and Dehlen call for further studies in that respect. Hutchinson *et al.* [11] reports on a survey of 250 professionals about the state of MDE in industry. A large number of these surveyed professionals found MDE advantageous in terms of productivity, maintainability and portability. However reported drawbacks include the need to learn new approaches and the cost of specialized tools. A survey of 155 Italian software practitioners on MDE presented in [3] concludes that modelling, particularly for code generation, is a well-established practice among the surveyed practitioners. Benefits of MDE were found to include support in design definition, improved documentation, easier maintenance and quality; while the main drawback is the effort involved in creating models. Other reviews on MDE adoption include [12], a case study that found that contextual forces dominate cognitive issues in using MDE, and [13] that reports on an experiment where code generation from models is compared to manual coding. Another group of related publications concern evaluations of the UML (e.g. [4] [14] [15]).

Unlike the related work described here, our work focuses on a particular MDE approach. We attempt to assess particular aspects of this approach in order to provide arguments in favour of its adoption. Further studies will be needed to evaluate the effectiveness of the approach in its practical use.

3. A Model-Driven Web-Engineering Approach (MODEWIS)

Figure 1 shows an overview of the MODEWIS approach. The approach is based on the MDA distinction between CIM, PIM, and PSM. We further distinguish two levels of abstraction within the PSM: an Abstract-Plat-

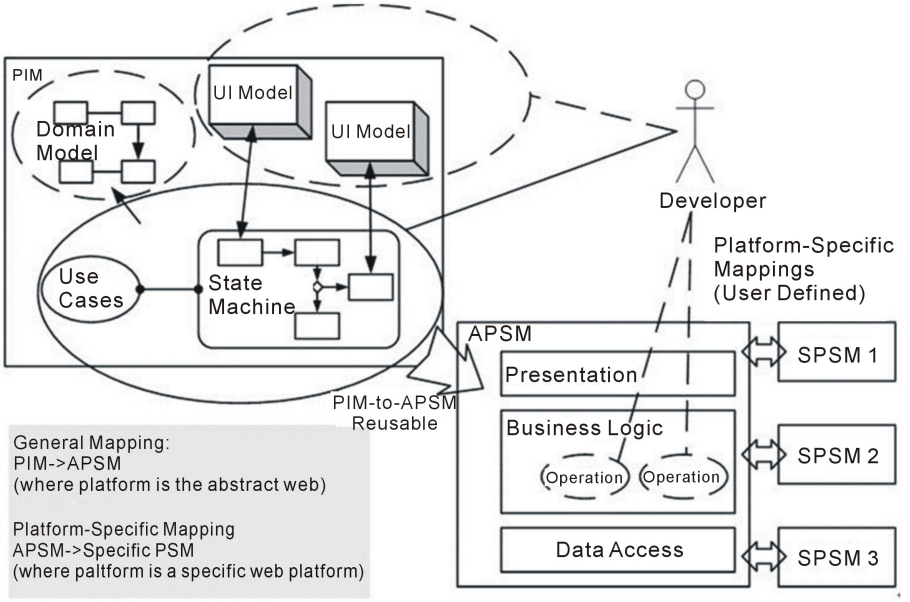

Figure 1. Overview of MODEWIS approach.

form Specific Model (APSM) and a Specific-Platform Specific Model (SPSM). Despite their variety, web technologies share many characteristics. These characteristics address web-application structure, navigation paths, logic control mechanisms, data access techniques, user interface (UI) components and the principle of separation of concerns. The APSM encompasses the common web technology characteristics, while the SPSM provides the specificity of concrete implementation platforms.

Figure 2 shows a high-level conceptual view of the APSM. The complete meta-model is described in [16].

The APSM is based on established web architectural patterns such as the Model-View Controller (MVC) pattern [17]. Elements of packages *State Machines*, *Communications* and *Use Cases* are mostly inherited from the UML specification [18]. The package *Domain* contains metaclasses required to define data entities, data composites and their attributes. The package *Service* contains metaclasses required to build data-access operations; these are dependent upon *Domain* package to perform operations. The *Controllers* package includes classes needed to manage both the change of status and the required low-level operations.

The APSM is platform-specific in the sense that it describes features specific to the web platform; it is also abstract [19] since it does not contain details of specific web platforms but only their shared features. SPSMs pertain to concrete web implementation platforms. We developed specific platforms corresponding to widely used technologies such as AndroMDA [20], WebRatio [21], Google Web Toolkit (GWT) [22] and Microsoft. Net [23]. A detailed description of these SPSMs is provided in [16].

The PIM is a subset of the APSM related to actors, use cases, state machines, presentation states and UI components such as Submit Buttons, Text and Images. **Figure 3** shows the elements of the APSM used as meta-model for the PIM. We developed a Domain Specific Language (DSL) for the specification of instances of the PIM. This DSL provides a visual notation for the specification of web-applications using state machine symbols from UML standard, as well as custom symbols for the description of user interface and data operations. A description of the DSL is provided in [16] and **Figure 4** shows a sample PIM.

Each use case is modelled as a UML state machine with presentations and data actions associated to states. The involved actor in **Figure 4** is *Member*. The use case involves only one state called *View Bill*. The UI presented in that state is enclosed within a group titled *View Bill*—to indicate the UI prototype of state *View Bill*. The presentation involves one data operation. In this operation, an instance of the entity *Bill* is retrieved to populate the UI components *Name*, *Address*, *Tel* and *Description* enclosed in a group named *Bill*. The operation uses the value found in a page variable *selected Bill* to filter the results of the query from the entity *Bill*. The page variable *selected Bill* is assumed to have been set in a previously executed use case.

Model transformation is central to a Model-Driven Development approach. The OMG [8] defines a transformation as an execution of a mapping. A mapping is a specification describing how to transform a PIM to PSM.

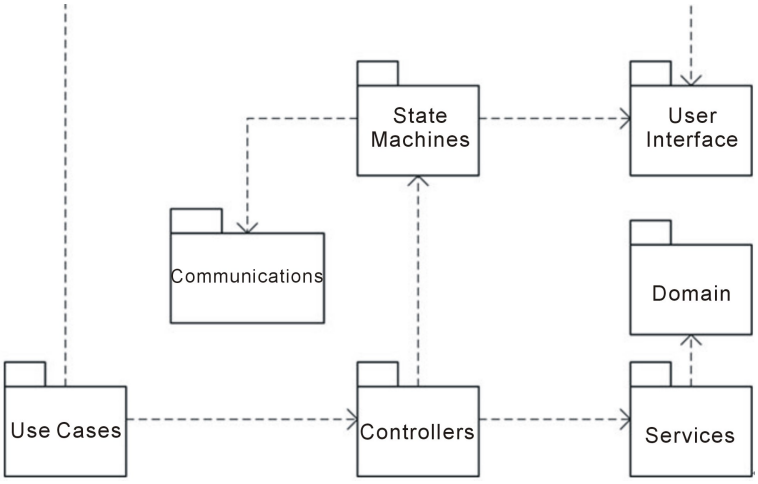

Figure 2. High-Level Package view of the APSM.

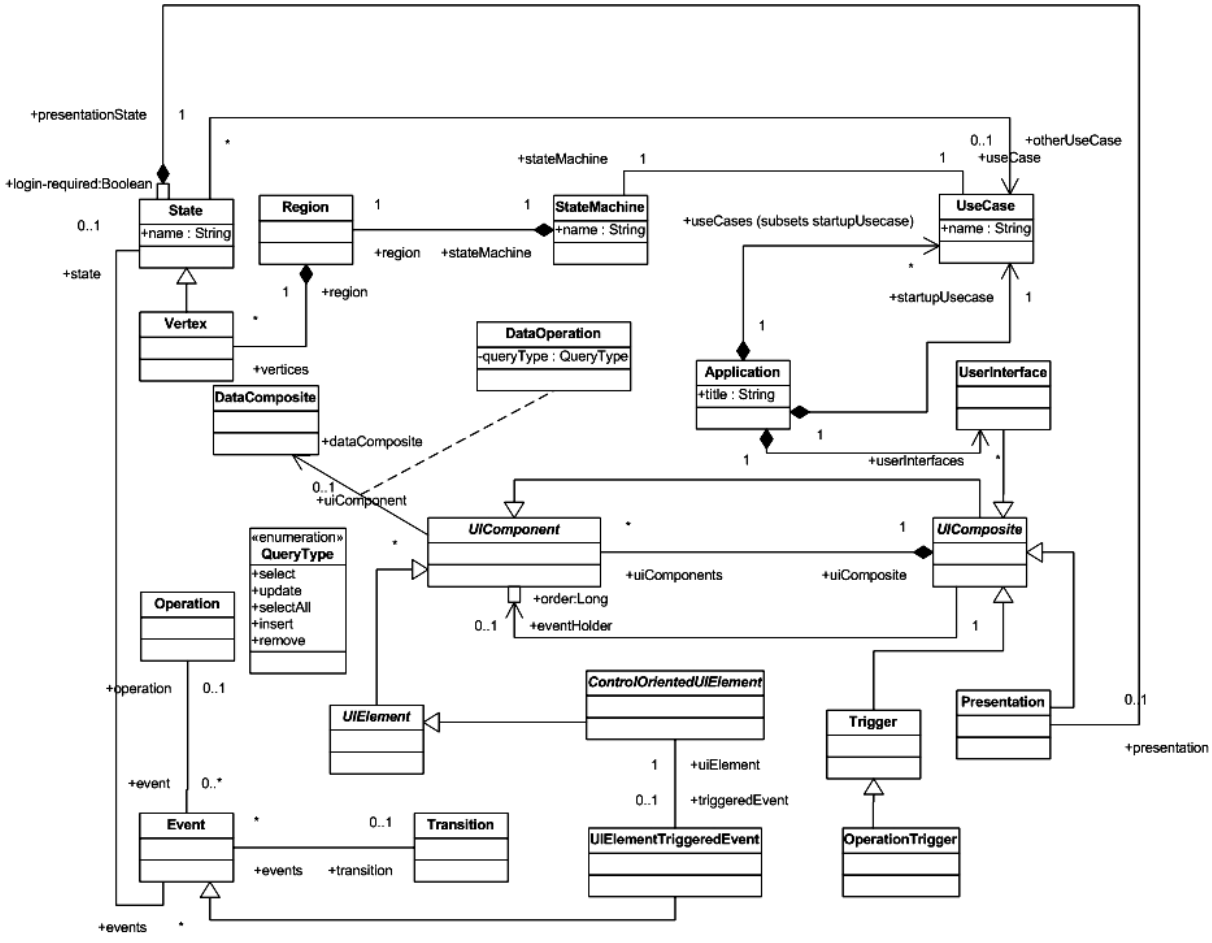

Figure 3. Part of the APSM meta-model used as PIM meta-model.

Our PIM-to-APSM transformations belong to the category of refining mappings according to Mellor *et al*. [24]. Refining mappings are defined between two sets of models both defined based on the same meta-model. The mapping refines a source model by adding more details to it. **Figure 5** presents a summary of the information used from the PIM as well as those created at the APSM level in a typical PIM-to-APSM mapping. Data objects

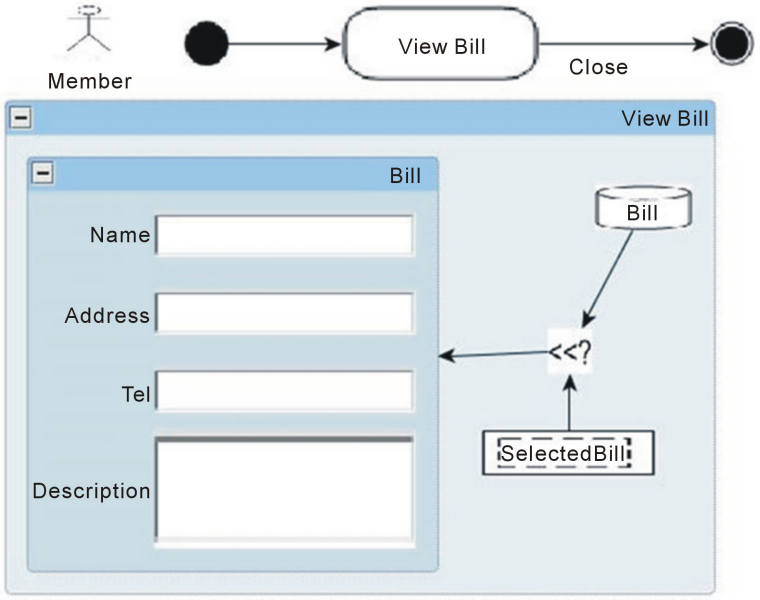

Figure 4. Sample PIM for use case view bill.

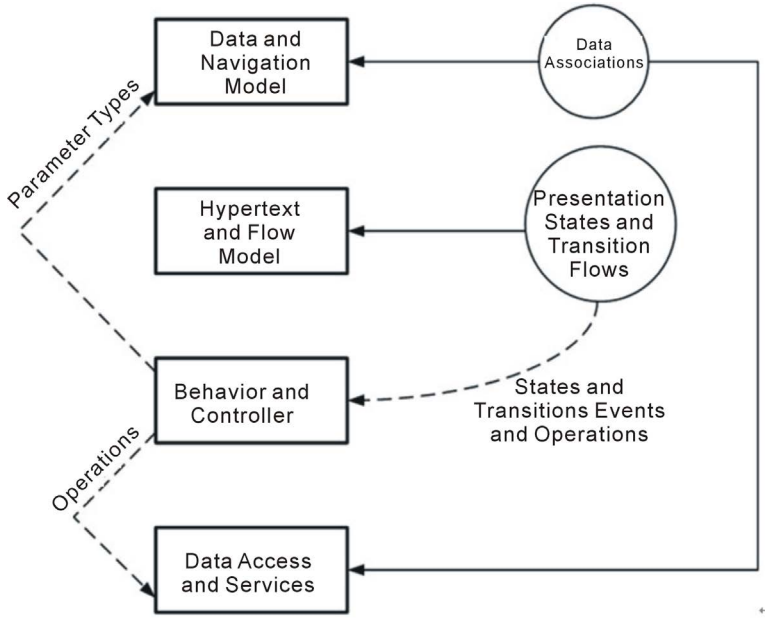

Figure 5. PIM-to-APSM mappings at a glance.

and navigation through data objects are created based on data associations found within the UI model. The contents and the structure of web pages are determined by the contents of the presentation states and transition flows. Transitions and states from the input model are also used to create events and operations used for generating the behavior and controllers of the application. The behavior and controllers are used in turn to build the data access services in combination with data associations from presentation states.

The transformations discussed above have been implemented as QVT relations [25]. The details of which can be found in [16]. As an example, the PIM modelled in **Figure 4** would result in the instance of APSM described in **Figure 6**. The generated APSM *has* an additional state (*Run State* 1) to handle the logic required to populate the billing details. From this state, a transition invokes operation *populate Bill Panel* in the controller class *View Bill Controller* which, in turn, calls *load Some* service operation from the service class *Bill Service*.

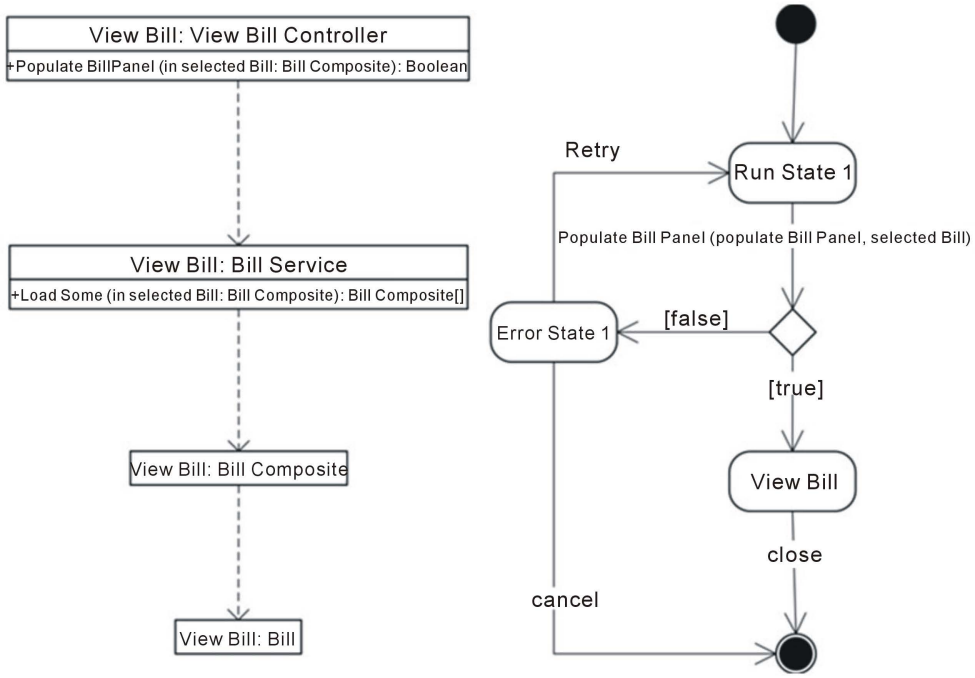

Figure 6. The APSM of the use case view bill.

The software development process put forward by MODEWIS approach consists on the following activities:

1) *PreConditions*: Requirements are elicited but not necessarily modelled. Use cases are realized and are optionally documented as textual descriptions. Data and usability requirements are considered so the developer can decide which data is used in every use case step as well as what is presented in every presentation step.

2) *PIMing*: A PIM serves as the input to the process and is composed of use case state machines and UI prototypes. This step is mainly manual and involves using MODEWIS DSL notation.

3) *APSMing*: The PIM is mapped to an APSM using the PIM_APSM set of QVT relations.

4) SPSMing: The resultant APSM is mapped to one of the specific platform.

5) Coding: Code is generated using code generators from the SPSMs.

6) PostConditions: The generated code might need to be manually edited to account for specific aspects of the target platform.

4. Assessment against Conventional Software Engineering

We compare MODEWIS approach with the conventional software engineering approach. Our goal is to show that a MDE such as MODEWIS retains some similarity with the conventional approach (with which most practitioners are familiar), while providing automation to several activities. According to [26], a conventional software engineering process consists of the following activities:

- Requirements/Specification, which consists in gathering requirements, understanding the domain, differentiating *what*s from *how*s and enlisting things the User would expect the software system to do.
- Design, where design choices such as selected technologies and comprising subsystems are made in order to ensure the satisfaction of requirements.
- Modelling, this is about visually specifying the designed system and requirements using modelling languages. Use case modelling, domain modelling and structural modelling are examples of modelling activity.
- Programming, this is the act of writing the actual code that implements the application functionality.
- Quality Assurance (QA), this activity includes the validation of the implemented software application to check that it satisfies its requirements. Typical QA activities are testing and reviews.
- Deployment, this process consists in releasing the application to the customer, installing required components and giving required instructions for using it.
- Management, which relates the managerial aspects of the software development including cost estimation,

task planning and resource assignation.

In **Table 1**, we compare the activities in MODEWIS with the conventional software engineering activities.

Requirements are out of the scope of MODEWIS and is considered a prerequisite. More specifically, functional requirements, usability expectations and domain data must be determined before the process starts. The specification of functional requirements is partly covered by the manual specification of state machines as part of the PIM.

Design affects our approach especially in terms of the architecture as well as the selection and setting up of the specific platform(s). APSMing corresponds in part to the high-level design (architecture) activity in the conventional approach as the PIM-to-APSM mappings integrate well established architectural patterns. The APSM-to-SPSM mappings consist in low level refinements to specific platforms (frameworks, databases...). The specific platform may or may not be among the four examples that we have implemented so far. If the specific platform is not available, the specific platform and its related APSM-to-SPSM transformations need to be defined based on the user's selection of implementation technologies. The up-front effort needed to develop these APSM-to-SPSM transformations is non-negligible; however they are re-used for any subsequent project involving the same platform.

Modelling with MODEWIS consists in defining the PIM that serve as the input to our approach. The rest of the modelling is performed in the context of automated transformations that create the APSM and relevant SPSMs. In conventional approaches the transformations PIM-to-APSM-to-SPSM correspond to the manual design refinements steps. In Section 6, we provide an assessment of the degree of modelling effort automation achieved through the PIM-to-APSM transformations.

Programming is supported in MODEWIS by the automated generation of the executable application code using code generators. This involves all data processing operations and many other logical operations performed for checking the validity of the input and perform web-specific tasks such as sending emails and comparing input values. Some manual coding may be required for complex business logic, look-and-feel as well as to improve performance.

Quality Assurance (QA) is performed against the models and transformations. In a conventional software engineering process, the focus of quality assurance is mostly on the quality of code. MDE approaches focus on models and transformations. QA is shifted to ensuring the PIM correctly captures the requirements and that transformations themselves are sound. MODEWIS uses a review approach based on proof-reading of the models.

5. Assessment of Design Inference

One of the prevailing tasks in software design is the refinement of abstract representations closer to the problem space, as more concrete models closer to the implementation. An effective model-driven approach must provide some automation to such refinement tasks by encompassing the capability to infer more detailed elements from abstract models. In order to assess this aspect of MODEWIS approach, we compare the operation refine-

Table 1. MDWE steps vs. conventional software engineering activities.

Conventional Activities	MDWE Steps					
	Pre	PIMing	APSMing	SPSMing	Code Generation	Post
Requirements/ Specification	Use case modelling as state machines					
Design			Architectural Patterns	Specific Platform definition		
Modelling		UI Prototyping and Use Case modelling using state machines	Automatically Done	Automatically Done		
Programming					Automatically Done	Manual coding as required
Quality Assurance		Proof-reading Transformation Results	Proof-reading Transformation Results			

ment capability with contracts modelling in Larman's approach [27]. In this approach, operations identified from use case descriptions are refined as contracts. These contracts provide a basis for creating sequence diagrams in subsequent steps of the development process. Our comparison is based on the *NextGen POS System* which is used as a main example in [27]. The *NextGen POS System* aims to assist cashiers in a department store. The main scenario of the use case *Process Sale* is presented in **Listing 1**.

In Larman's approach, software designers identify a set of *system operations* from the scenario and based on a domain model, specify contracts for each of these operations. An operation contract includes different sections, the most essential of which is the operation's postconditions stating the operation effects in term of changes to the domain objects, attributes and associations.

Table 2 shows the system operations and the postconditions contracts that Larman's approach suggests for use case *Process Sale*. These postconditions assume the prior development of a domain model with the classes and associations referred to. **Figure 7** shows the main scenario of use case Process *Sale* in our PIM DSL. We also assume a prior development of the system's domain model with classes corresponding to the data entities referred to in the PIM. **Table 2** shows the operations derived from the transformation of the PIM to an APSM. These operations are created for the controller of the use case *Process Sale*. Corresponding operations are generated for all *system operations*. Although some of Larman's *system operations* are fulfilled by several operations in MODEWIS approach. All the operations post conditions as specified by Larman's are fulfilled as follow:

- *makeNewSale*: *loggedRegister* is considered an attribute of the data variable *newSale*. Hence, *newSale* is associated with *loggedRegister*. This satisfies postconditions 11, 12 and partly 13. The rest of the attributes of *newSale* are set in the context of other operations and use cases.

- **Customer arrives at a POS checkout with items to purchase.**
- **Cashier starts a new sale.**
- **Cashier enters item identifier.**
- **System records sale line item and presents item description, price, and running total.**
- **Cashier repeats steps 3-4 until indicates done.**
- **System represents total with taxes calculated.**
- **Cashier tells customer the total and asks for payment.**
- **Customer pays and System handles payment.**
- **The Cashier tells the customer the total.**
- **System logs completed sale and sends sale and payment information to the external Accounting system.**
- **System presents receipt.**
- **Customer leaves with receipt and goods**

Listing 1. Main Scenario of use case process sale.

Table 2. Larman's contract vs. MODEWIS generated controller operations.

Larman's approach		MODEWIS
Operation Contract	Postconditions	Controller Operations
1—makeNewSale	11—A Sale instance s was created 12—s was associated with a Register 13—Attributes of s were initialized	setNewSale(newSale, loggedRegister)
2—enterItem	21—A SalesLineItem instance sli was created 22—sli was associated with the current sale 23—sli.quantity became quantity 24—sli was associated with a ProductDescription based on itemID match	populateItemsTable(id, newItem) setNewItem(newItem) addToItemsTable(newItem) updateNewSale(newItem, newSale)
3—endSale	31—Sale.isComplete became true	setIsComplete(newSale, "true") updateSale(newSale, isComplete, newPayment, items, subtotal, tax, balance)
4—makePayment	41—A Payment instance p was created 42—p.amountTendered became amount 43—p was associated with the current Sale 44—The current Sale was associated with the Store	handlePayment(newSale, items)

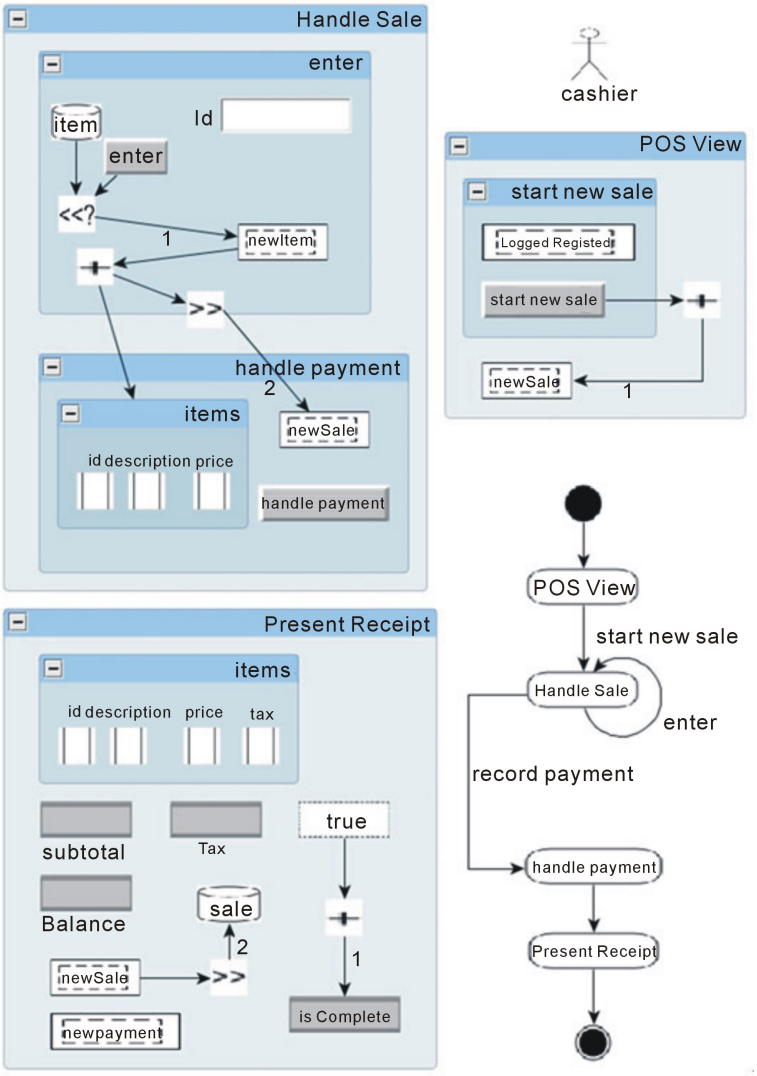

Figure 7. The PIM of use case process sale.

- *enterItem*: Operations *populateItemsTable* and *setNewItem* handle postconditions 21 and 22. The operation *populateItemsTable* also satisfies the postcondition 24 because it loads data based on the input *id* from the data table *item*. Also having table columns *description* and *price* associates these attributes with *id*.
- *endSale*: This is handled by two operations: *setIsComplete* and *UpdateSale*. Note that the operation *update-Sale* accepts the values of all components although the values of *subtotal*, *tax* and *balance* exist in the variable *newPayment* as well. Currently we have no mechanism to detect and remove this type of dependency.
- *makePayment*: The operation, *handlePayment* accepts the parameters *newSale* and *items*. Since handling payment is considered a separate use case, the output that is *newPayment* is inserted in the final presentation unit *Present Receipt*. The operation *updateSale*, which runs at the beginning of loading this state, confirms that *newPayment* should also be assigned as an attribute association with *newSale*.

6. Productivity Assessment

We quantify the degree of modelling effort provided by MODEWIS approach by considering the number of model elements automatically generated in models. In MODEWIS, software is first modelled as a PIM in which elements are automatically added to form an APSM. We can estimate the degree of automation by considering the ratio of automatically added elements versus the provided input elements. In a manual approach, developers

Table 3. Degree of automation results for dilmaj and AMS case studies.

Case Study	Degree of Automation results		
	Number of PIM Elements	Number of APSM Elements	APSM vs PIM
Dilmaj	588	1509	61%[a]
AMS	858	3646	76%

[a]61% of the APSM are automatically added elements to the PIM.

Table 4. Proportion of generated code for the AMS case study.

APSM vs. PIM	AndroMDA vs. APSM	GWT vs. APSM	WebRatio vs. APSM	.Net vs. APSM
76%	33.33%	36.78%	46.77%	59.78%

would have manually added these elements as part of model refinement.

We evaluated the degree of automation of modelling effort obtained from our approach by examining two case studies: Dilmaj [28] a Web 2.0, collaborative, multilingual project about languages and language development supported by a group of developers, and Account Management System (AMS) [29], a commercial project foreseen by a waste/water management company to replace an existing non-web system. **Table 3** shows the percentage of added elements in the APSM versus those of the PIM for the case studies *Dilmaj* and *AMS*.

The MODEWIS approach automatically generates SPSMs from APSM and subsequently, code from the SPSMs. **Table 4** shows the proportion of additional model elements generated with respect to the *APSM* for the AMS case study in each of the SPSM platforms: *Andro MDA*, *GWT*, *Web Ratio* and *.Net*.

7. Conclusion

The adoption of MDE approaches is still lagging despite their many promises. In this paper we presented an assessment of MODEWIS a MDE approach for web-applications development. We compared the MODEWIS process with the conventional Software Engineering process. This showed that MODEWIS is not "revolutionary" but rather "evolutionary" as the MDE process still aligns with the conventional process. We also looked at the automated design inferences compared to manual design decisions and the value added by automated transformations to models. This assessment is a first attempt at presenting arguments in order to alleviate practitioners' scepticism toward a MDE approach such as MODEWIS. The conducted case studies offer some other insights into the value of Model-Driven Development that would need further evaluation. For instance, the Dilmaj development team had estimated that the amount of time required to manually develop a working prototype of the system was about three months. Using MODEWIS, we were able to develop a working prototype in one week. However, although the resulting application is completely functional, optimizations to data queries as well as the appearance of the UI were considered by developers as changes required to be done to the automatically generated application in order to complete the system. As future work we plan to conduct an evaluation in order to assess the global productivity gain of MODEWIS in regard to these needs for changes to generated code.

References

[1] France, R. and Rumpe, B. (2007) Model-Driven Development of Complex Software: A Research Roadmap. *Future of Software Engineering* (*FOSE* '07), 37-54.

[2] Kleppe, A.G., Warmer J.B. and Bast, W. (2003) MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc.

[3] Torchiano, M., Tomassetti, F., Ricca, F., Tiso, A. and Reggio G. (2013) Relevance, Benefits, and Problems of Software Modelling and Model Driven Techniques—A Survey in the Italian Industry. *Journal of Systems and Software*, **86**, 2110-2126. http://dx.doi.org/10.1016/j.jss.2013.03.084

[4] Petre, M. (2013) UML in practice. *Proceedings of the* 2013 *International Conference on Software Engineering* (*ICSE* '13), San Francisco, 18-26 May 2013, 722-731. http://dx.doi.org/10.1109/ICSE.2013.6606618

[5] Selic, B. (2012) What Will It Take? A View on Adoption of Model-Based Methods in Practice. *Software and System Modeling*, **11**, 513-526. http://dx.doi.org/10.1007/s10270-012-0261-0

[6] Selic B. (2006) Model-Driven Development: Its Essence and Opportunities. 9*th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing* (*ISORC* 2006), Gyeongju, 24-26 April 2006, 313-319.

http://dx.doi.org/10.1109/ISORC.2006.54

[7] Fatolahi, A., Somé S.S. and Lethbridge, T.C. (2011) Model-Driven Web Development for Multiple Platforms. *Journal of Web Engineering*, **10**, 109-152.

[8] Object Management Group (OMG), MDA® Specifications. http://www.omg.org/mda/specs.htm

[9] Fatolahi, A., Somé S.S. and Lethbridge, T.C. (2012) A Meta-Model for Model-Driven Web Development. *International Journal of Software and Informatics, Special Issue on Software Modeling*, **6**, 125-162.

[10] Mohagheghi, P. and Dehlen, V. (2008) Where Is the Proof?—A Review of Experiences from Applying MDE in Industry. *Lecture Notes in Computer Science*, **5095**, 432-443.

[11] Hutchinson, J., Whittle, J., Rouncefield, M. and Kristofferson, S. (2011) Empirical Assessment of MDE in Industry. *Proceedings* 33*rd International Conference on Software Engineering* (*ICSE*'11), Honolulu, 21-28 May 2011, 471-480.

[12] Kuhn, A., Thompson, A. and Murphy, G. (2012) An Exploratory Study of Forces and Frictions Affecting Large-Scale Model-Driven Development. *Lecture Notes in Computer Science*, **7590**, 352-367.

[13] Papotti, P.E., do Prado, A.F., Lopes de Souza, W., Cirilo, C.E. and Pires, L.F. (2013) A Quantitative Analysis of Model-Driven Code Generation through Software Experimentation. *Lecture Notes in Computer Science*, **7908**, 321-337.

[14] Arisholm, E., Briand, L.C., Hove, S.E. and Labiche, Y. (2006) The Impact of UML Documentation on Software Maintenance: An Experimental Evaluation. *IEEE Transactions on Software Engineering*, **32**, 365-381. http://dx.doi.org/10.1109/TSE.2006.59

[15] Briand, L., Labiche, Y. and Madrazo-Rivera, R. (2011) An Experimental Evaluation of the Impact of System Sequence Diagrams and System Operation Contracts on the Quality of the Domain Model. *Proceedings International Symposium on Empirical Software Engineering and Measurement* (*ESEM* 2011), Banff, 22-23 September 2011, 157-166. http://dx.doi.org/10.1109/ESEM.2011.24

[16] Fatolahi, A. (2012) An Abstract Meta-model for Model Driven Development of Web Applications Targeting Multiple Platforms. Ph.D. Dissertation, University of Ottawa, Ottawa. www.ruor.uottawa.ca/en/handle/10393/23262

[17] Reenskaug, T. (1979) MODELS—VIEWS—CONTROLLERS. Technical Note, Xerox PARC. http://heim.ifi.uio.no/~trygver/mvc/index.html

[18] Object Management Group (OMG) (2013) Unified Modeling Language (OMG UML). http://www.omg.org/spec/UML/2.5/Beta2/PDF/

[19] Almeida, P.J., Dijkman, R., van Sinderen, M. and Pires, L.F. (2004) On the Notion of Abstract Platform in MDA Development. *Proceedings of the* 8*th IEEE International Enterprise Distributed Object Computing Conference* (*EDOC* 2004), Monterey, 20-24 September 2004, 253-263.

[20] AndroMDA. http://www.andromda.org

[21] WebRatio. http://www.webratio.com

[22] Google Web Toolkit, Google Code. http://code.google.com/webtoolkit/

[23] Microsoft. Net Framework. http://www.microsoft.com/net/

[24] Mellor, S.J., Scott, K., UHL, A. and Weise, D. (2004) MDA Distilled: Principles of Model-Driven Architecture. Addison-Wesley, Boston.

[25] Object Management Group (OMG) (2008) MOF QVT Specification. http://www.omg.org/spec/QVT/1.0/PDF/

[26] Lethbridge, T.C. and Laganière R. (2001) Object-Oriented Software Engineering: Practical Software Development Using UML and Java. McGraw-Hill.

[27] Larman, C. (2005) Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Prentice Hall PTR.

[28] Dilmaj. http://sokhangozaar.appspot.com/?locale=en_US#

[29] AMS_APSM. http://www.site.uottawa.ca/~afato092/AMS_APSM_Ali_Fatolahi.zip