A New Framework for Classifying Information Systems Modelling Languages

Ahmad F. Subahi*, Youseef Alotaibi
Department of Computer Science, University Collage of Al-Jamoum, Umm Al-Qura University, Makkah, Saudi Arabia.

* Corresponding author. Tel.: +966550055856; email: AFSubahi@uqu.edu.sa
Manuscript submitted October 5, 2017; accepted December 1, 2017.
10.17706/jsw.13.1.18-42

Abstract: The comparative survey shows that there are many Domain Specific Languages (DSMLs) adopted in various Model-Driven Engineering (MDE) approaches for information systems engineering. Choosing the appropriate DSML depends on the type of the target information system (IS), as well as the technical and modelling skills of the developing team. Adopting the right DSML is considered a difficult issue for casual designers, domain experts with limited modelling skills. Therefore, introducing a simplified DSML that describes IS in a higher-level of abstraction way than existing DSML approaches supports business users to contribute more in the development process. In this work, a classification framework is introduced to categorise thirteen DSMLs, based on the target type of IS, into four major categories, namely, (1) DSMLs for n-tier web application development, (2) DSMLs for cloud-based applications development, (3) DSMLs for mobile-based applications development and (4) DSMLs for tier-specific enterprise applications development. Feature modelling technique was adopted to compare and categorise modelling languages. Sharing a common similarity are clustered together into higher-level categories in the hierarchy as the main characteristics from different modelling languages. At the end, the resulting framework is documented using a multiple-level feature diagram to support domain expert decisions of choosing a DSML that suits their needs; And provide common DSML features to be adopted in constructing a simplified DSML that supports Lightweight MDE development.

Key words: Domain-specific modelling languages, end-user development, information systems engineering, lightweight model-driven engineering.

1. Introduction

Model-driven Engineering (MDE) is a software development methodology that uses models as primary artefacts in the development lifecycle rather than executable code. Models are considered system abstractions that allow developers to effectively describe the structural and behavioural aspects of a system. The OMG Model-Driven Architecture (MDA), Microsoft Software Factories and Model Integrated Computing (MIC) are well-known examples of the MDE initiatives appeared in the last decade, which aim at raising the abstraction level of development using models. One of the main part of any Model-driven development approach is the modelling language. Many modelling languages have been proposed for modelling software systems both in a particular domain and as general-purpose modelling languages. These languages are mainly used for describing various system concerns at different levels of abstraction.

This survey attempts to present a definition framework that is used to classify and differentiate between several existing modelling languages. According to the review of the related literature on the recent MDE

approaches and tools for information systems engineering, the Domain-Specific Languages adopted in these approaches are categorised, mainly, into three groups: (1) DSLs for web applications development, such as n-tier Web Modelling Language (WebML) [1] and Web Domain-Specific Language (WebDSL) [2]. (2) DSLs for cloud-based applications development, such as, Mobile Cloud DSL mobiCloud [3]; and (3) DSLs for mobile applications development, such as Mobile DSL mobiDSL [4].

It is worth mentioning that there are two additional groups can be also considered in the classifications presented in this survey: (4) DSLs for tier-specific development, such as Sculptor [5] for establishing a (visualised) persistent data store, or database, tier, UsiXML [6] for constructing a presentation tier (GUI) and as Object-Process Methodology (OPM) [7] for developing embedded systems (devices) that works with information.

The main contribution of this work is investigating common key features appear in well-known domain-specific languages (DSLs) to introduce a simplified business user-friendly modelling language. This language must cover crucial aspects appear in various types of information systems, to be used in Model-Driven End-User Development for Information System Engineering (MDEDIS). The promising development approach can be used to support business end-users, who have limited modelling skills, working in a more rational way and contributing more, via the utilisation of a domain-specific modelling language that suites their skills and serves adequately their needs, during the development process.

This paper is organised as follows; section 2 presents brief overview of the fundamental principles of MDE with examples, namely Model, Metamodel and Modelling language. Section 3 discusses the related works that have been done on the domain of classification of modelling languages. Section 4 highlights the adopted methodology for conducting the survey on existing DSMLs. Section 5 describes four major categories of DSMLs that are used for developing information systems. For each category, a number of DSMLs are covered. Section 6 introduces a framework for modelling languages classification that is based on the four major categories of DSMLs, presented in section 5. Section 7, presents complete analysis of the data that is generated from DSMLs comparisons. Section 8, provides some recommendations that can be adopted for developing a lightweight DSMLs for business end-users. Section 9, presents a conclusion of the research survey paper.

2. Core Principles of Model-Driven Engineering

2.1. What Is Model?

In the context of Software Engineering, a model is considered a simplified descriptive representation of a SUS (System Under Study) that might be an existing system or a new one under development [8]. It describes software system structure, operations and interrelationships between its components. These aspects, captured from reality, are represented as a set of words and/or symbols to enable creating and managing models or diagrams of that system [9], [10].

There are various utilisations of models in software engineering. According to [11], models have many contributions during software project lifecycle. Models can facilitate the communication between stakeholders and developers team, and allow sharing a common vision and technical and/or non-technical knowledge between them [12], [13]. Models also provide more appropriate abstract views of the SUS, which leads to achieve project control during the development process. Additionally, models serve as a design specification for a software system [8], or serve as the basis for a code generation stage in MDE approaches [10], [14].

2.2. What Is Metamodel?

According to the OMG standards [8], [15], a metamodel is a specification model that defines a valid model

expressed in a particular modelling language (a model of a model). Every model must conform to a metamodel in which all acceptable structure of and relationships among model elements are specified in the metamodel. Therefore, any instance model must meet these constraints and satisfies these specification rules to be considered valid and well-formed [8]. There are many widely-used examples of metamodel representations, such as Backus-Nuer Form (BNF), grammar or context free grammars, UML Profiles or MOF.

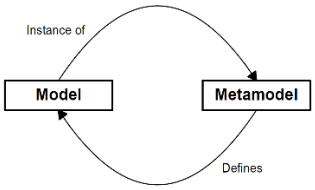

Fig. 1. The relationship between models and metamodels

2.3. What Is Modelling Languages?

According to [16], A modelling language can be defined as a set of all possible models that are conformant to an abstract syntax, expressed by one or more concreate syntaxes to satisfy certain semantics [67][68]. The modelling language is created from a metamodel within a certain domain. The abstract syntax of the language is defined and validated in the metamodel of the modelling language, while the concreate syntax is presented in the models of that language.

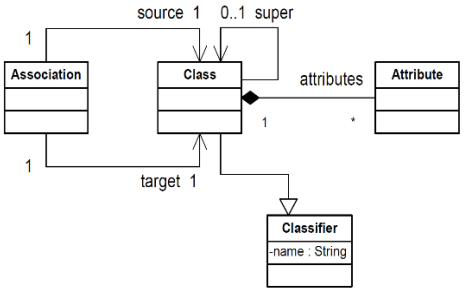

Fig. 2. An example of models.

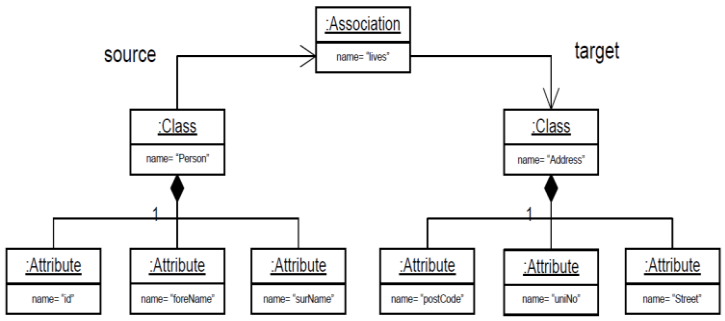

Fig. 3. An example of metamodels.

Fig. 2 demonstrates a portion of the UML metamodel. It shows how language concepts at the metamodel layer, such as Class, Attribute and Association, are related and defined. On the other hand, Fig. 3 illustrates how the user designed elements, that conforms to the metamodel in Figure 2, are defined.

3. Related Work

There are several works have been done on modelling languages classification to support developer choices of modelling languages. Each classification system focuses on a particular domain problem, such as business process, machine learning and model transformations. For example, SEQUAL is a framework that was presented in 1994 with the goal of evaluating systematically the quality of information systems and other conceptual models via the notions of syntactic, semantic and pragmatic applied to them. This framework was improved many times later to cover more evaluating features. SEQUAL's formulas are based upon the audience, the language, the model, the domain, the audience's knowledge and the audience's interpretation [17].

This framework is applied to compare some Modelling Languages that could be used for business process modelling. The three selected BPM languages were EEML, UML and BPMN including the following items:

Goals of modelling task, language extension, domain, externalized model, knowledge of the stakeholders, the social actors' interpretation and the technical actors' interpretation [17][66].

Furthermore, Czarnecki and Helsen in [18], also propose a framework for the classification of several existing and proposed model transformation approaches. They expressed their classification system using feature model that makes the various possible design choices for a model transformation strategy explicit. Examples of approaches, for each of the presented design choices, are proposed instead of going deeply in the details of each individual approach. This framework is applied on the major model transformation categories, namely, model-to-model transformation and model-to-text generation.

Additionally, the ARENA Framework is introduced to enlighten individuals that work with models on this area. The main goal of ARENA is to help the user choosing the most appropriate modelling language, in order to assure a quality output, to feel more supported and to work in a more rational way. The framework was applied in [11] to evaluate and compare the quality of User-Interface Modelling Languages (UIMLs), focused on either their general and specific characteristics. All domains share the General Properties, as opposite to Specific Properties, that differ from each other [11].

Moreover, a survey on DSMLs/DSLs for implementing machine learning algorithms for big data applications is presented. It utilises various classification features used for distinguishing and comparing critical seven DSLs that are used specifically in that domain [65]. This supports language engineers to implemented algorithms using more informed choices. Properties considered in their classification system involved generic features, such as representation styles (declarative/imperative), development stages (requirement/programming/modelling) and target platforms. Additionally, more specific features related to machine learning domain are also considered, such as supporting vector/mitrex/ graph and supporting distributed computing/cloud computing.

In contrast to the related work considered in this research [11][17][18][65], our work focuses on the information system engineering domain. It aims at determining critical information system features to be used later in abstracted way for introducing a simplified business user-friendly modelling language. We bring different types of DSLs/DSMLs together in a classification framework to investigate common information system features captured by several modelling languages. The target types of systems are mainly considered in this classification system. This strategy is essential to cover a wide range of domain specific aspects from various kinds of information systems.

4. Methodology of Literature Review

The literature survey is conducted systematically on a series of peer-reviewed academic publications of MDE approaches, tools and techniques, including their adopted modelling languages (DSMLs) over the past 14 years, from 2003 to 2017. As MDE and DSML are considered active research topics in Computer Science and Software Engineering, massive number of works regarding their development approaches, and modelling languages design and implementations have been introduced and published during the last two decades. Thus, an efficient method to process this large amount of literature must be established. A two-stage approach for reviewing the literature is applied to the work presented in this survey (Figure 4) [19], [20].

In the first stage, a comprehensive search of relevant publications is applied using keywords, such as "domain-specific language for SAAS applications", "domain-specific language for cloud applications development", "DSL for mobile applications development", "model-driven development framework for generating web applications", "MDE approach for information systems engineering", "end-users MDE approach for generating information systems" and "eclipse graphical editor for DSL". Two filtering steps, then, are applied to the search results. The retrieved publications are filtered first based on their titles. After

that, abstract of the filtered research papers are read to examine whether each publication satisfies our concern or not. As an outcome, related literature of MDE approach and modelling languages are identified.

In the following stage, the conclusions of the selected papers and the main parts of their full text are read and reviewed. As a result of these filtering steps, the most relevant literature is chosen and selected to be involved in the survey presented in this paper.

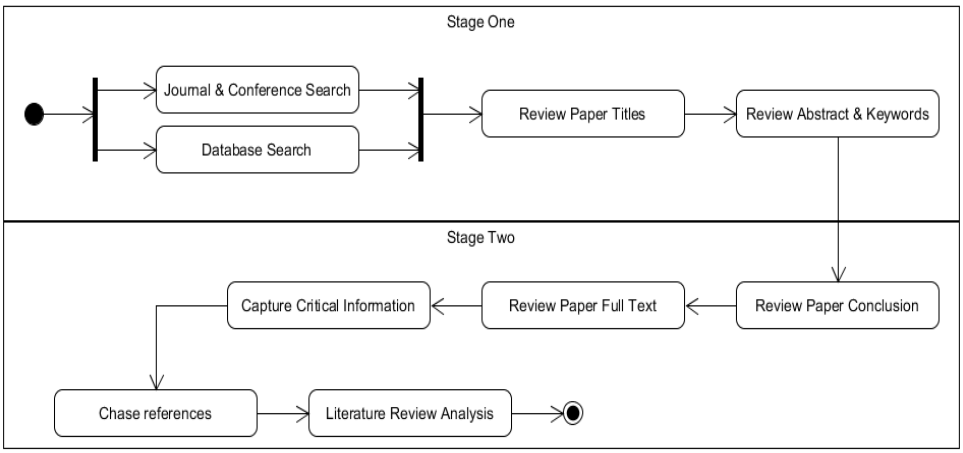

Fig. 4. The Methodology of the literature review.

5. Modelling Languages Classification

We have classified the domain-specific modelling languages (DSMLs), in the literature survey, into four major categories, namely, DSMLs for n-tier web application development, DSMLs for cloud-based applications development, DSMLs for mobile-based applications development and DSMLs for tier-specific enterprise applications development. The following subsections represent a selection of common DSMLs that fit in the proposed DSML categories.

5.1. DSMLs for N-tier Web Applications Development

5.1.1. WebML

WebML is a DSML for specifying the conceptual model of J2EE web applications. It is supported by a CASE tool, WebRatio for enabling complete code generation. The WebML metamodel consists of various graphical views to allow expressing all critical concepts of the domain of web engineering, such as data, service, navigation, and processes. The language has several basic views that represent the overall structure of web applications, namely, data, hypertext and presentation view [1]. These views are extended later to cover more views for representing web services and process workflow via service view and business process view respectively [1].

Based on the notation of the standard Entity-Relationship model (ER), the WebML data model is used for presenting the data schema including entities, attributes and relationships. Furthermore, the graph-like WebML hypertext model represents the navigation path between units within of a web application, such as views, areas, pages, and content, forming the overall structure of the domain. The widely-known notation of Business Process Modelling Notation (BPMN) is adopted to cover basic concepts for representing workflow in web application. The presentation model, additionally, specifies visual representations and styles of the pages on screen [1].

5.1.2. UML-RSDS

UML-RSDS (Reactive System Development Support) language is a subset of UML 2.0 with a precise semantics [21]. The semantics of the language is formalised via the axiomatic approach, using Real-time

Action Logic (RAL). The graphical notation of the language consists, mainly, of three types of UML 2.0 models, namely, Class diagram, State machine diagram and Use case diagram, whereas the OCL constraints language is used as a textual notation [22][23]. It can be used for specifying 5-tier Enterprise Information Systems (EIS), including Client, Presentation, Business, Integration and Resource tier [23].

Moreover, The UML-RSDS metamodel is implemented under Eclipse Modelling Framework (EMF) as several Ecore models. Its toolset supports the specification and analysis of systems in this subset, and the generation of executable code from specifications [21]. According to [23], the approach targets three kinds of EIS technology styles, namely, Servlet, JSP, and J2EE.

5.1.3. WebDSL

WebDSL is a Domain-Specific Language (DSL) for implementing dynamic web applications with a rich data model. It has a textual notation for modelling technical aspects, such as data model, web pages, principles (users), access controls and workflow. Each aspect is considered a WebDSL sub-language that might be a part of the core language or an extension. For example, the DSL workflow is a modelling language, an extension of the core WebDSL, for describing business workflows in web applications. Like other workflow modelling languages, the WebDSL workflow language define functions and procedures associated with business objects (entities). These operations reflect coordinated activities between different actors [2].

The WebDSL syntax is defined and formalised using Syntax Definition Formalism (SDF2). One benefit of SDF2 is the ability of integrating the definition of the lexical and context free syntax of the language. Additionally, SDF2, as a meta-syntax, can combine definitions for different (sub-)languages. WebDSL is implemented using Stratego/XT framework. Stratego/XT consisits of two major components (Stratego and XT). Stratego is a transformation language that is based on rule rewriting paradigm, whereas XT is a set of transformation tools that provides infrastructure of transformation systems including parsing [24], [25]. The targeted technologies of the generated web systems is Java as an implementation language, SQL as a database system, JFS presentation language with the EL expression language for accessing data and XML for configuration of frameworks [24].

5.1.4. Wamm UML

WAMM UML is a domain-specific modelling language for simplifying the construction of web applications using high level models from a domain structure viewpoint, rather than low level solution models. The starting point of the series of model transformations is a UML Domain Model, a profiled UML Class Diagram, that represents the basic structure and relationships between real-world objects in a domain of interest (business entities) [26].

The concreate syntax of the WAMM Domain Model is defined using a UML profile, for specialising UML metaclasses, called WebApp profile. The WebApp UML profile is implemented using Eclipse Modelling Framework (EMF). The model transformation engine of this approach, in addition, is implemented using ATL (Atlas Transformation Language), whereas a template code generation strategy, JET (Java Emitter Template, is adopted for generating the final code [26].

5.1.5. UWE

Like other MDA approaches, the modelling language in the Unified Web Engineering (UWE) approach is completely based on the UML 2.0 standards. Its metamodel is defined using a UML profile that provides a precise description of the concepts adopted for modelling web applications and their semantics. It consists of six major views related to the domain of web engineering, namely, requirements, content, navigation, process, presentation and adaptation view. The defined models of a web system are several stereotyped UML diagrams, such as Class, Use case, Activity and State Machine diagram [27].

The generated web applications are based on the Java Spring MVC architecture. Java Server Page (JSP)

and Java beans, as dynamic web page technologies, are supported in UWE. The development process of UWE is guaranteed via OpenUWE MDE framework that contains two main CASE tools, namely, ArgoUWE and UWEXML. The ArgoUWE help the design activities, in which it comprises all required UWE profiles implemented as a plugin module of the open source ArgoUML modelling tool. Besides, the UWEXML tool is utilised as a code generation framework for generating Web applications automatically [28].

5.1.6. OOWS

OOWS is a web development approach that tends to capture the web applications requirements using high-level conceptual models to generate operational prototypes from these specifications. The language of OOWS is considered a multiple view modelling language, in which it has two major web domain aspects (views) for describing specifications of a web system, namely, navigation, presentation, navigational context. Furthermore, OOWS modelling language also has classic Structural and behavioural UML models, adopted in the original OO-Method [29].

As the OOWS modelling language is UML-based, system concepts are defined using UML profiles. The adopted profile consists of several stereotypes, such as view and context. The context stereotype allows us to define a view over a set of stereotyped classes (views). These stereotypes and more specify n-tier web applications, including presentation, application and persistence tier [29].

5.2. DSMLs for Cloud-Based Application Development

5.2.1. MobiCloud

MobiCloud is a Ruby-based domain-specific language (DSL) for specifying cloud-based applications that run over mobile devices [3]. It is formalised using grammar approach via BNF specification language. MobiCloud offers a sufficient abstraction level that covers core cloud and mobile features. The language is capable to generate applications for different front-end mobile technologies, such as Android and Blackberry applications. Moreover, Google App Engine (GAE) and Amazon (EC2) are back-end cloud technologies adopted by MobiCloud. The language is supported by a graphical tool that can be used to generate code using graphical components, called MobiCloud Composer [3], as well as a web-based text editor with a Ruby compiler [30].

The Model-View-Controller (MVC) design pattern is the adopted pattern for describing cloud-mobile applications using MobiCloud modelling language. At the metamodel level, the language provides a construct for each MVC component to contain all critical detail for that component, namely, Model, View, Controller. The Model construct is utilised for expressing the data structure needed to store application data. In a different manner, the Controller generates, automatically, operations for creating and retrieving a task data structure, whereas the View provides a basic user interface to add and retrieve tasks. It worth mentioning that generated applications only support simple CRUD operations [3].

5.3. DSMLs for Mobile Applications Development

5.3.1. MobiDSL

MobiDSL is a lightweight domain specific language (DSL) that is used for identifying the core requirements and various design elements of mobile web domain. It has a textual notation that enables a developer to define the specification of mobile web applications at a very high level of abstraction, without requiring any knowledge of web programming. Specifically, the supported mobile web architecture by MobiDSL is a basic 3-tier RESTFUL architecture that employs simple CGI and SQL APIs with a simple HTTP protocol [4].

The widely-known XML markup language is used for defining the MobiDSL metamodel. Crucial mobile web application concepts are expressed via the metamodel, such as page structure, hypertext (navigation),

data retrieval, query result formatting, transactional logic and validation strategy. Small successful cases, from the real-world is provided in [4] showing the implementation of a mobile PHP web application with MySQL database.

5.3.2. XIS-mobile

XIS-mobile is a domain-Specific Language (DSL) for describing mobile applications in a platform-independent way and using domain specific concepts. Its MDA development approach aims at mitigating commonly-known software development challenges, in the context of mobile applications, namely, the software development complexity and the mobile platform fragmentation.

The abstract syntax of the language is defined using a UML profile. It consists of four main system views, namely, Entities, Architectural, UseCases and User-Interfaces. The Entities views is used for expressing concepts of the problem domain. It has two sub views, one for describing domain entities (Domain view) and another one for representing business entities as classes (BusinessEntities view). All these views are defined using the XisEntities and XisBusinessEntities stereotype respectively [16].

Furthermore, the Architectural view expresses the interactions between the mobile application and the external entities. The XisMobileApp stereotype is designed for defining concepts of this view. In addition, The UseCase view expresses the operations performed, by each actor, when interacting with the application. The UseCase might have two stereotypes, namely, XisEntityUseCase and XisServiceUseCase. Besides, the User-Interface view has two sub-views, the NavigationSpace and InteractionSpace view, that are used to define the screens of the mobile application and the navigation flow between them.

The language is implemented using two well-known technologies, the Sparx Systems Enterprise Architect (EA) and the Eclipse Modelling Framework (EMF). It is worth mentioning that the XIS-mobile helps the developer by generating the repetitive and boilerplate code of the generated mobile application only. The developer, then, can customise the generated code if it does not satisfy all their demands [16].

5.4. DSMLs for Tier-Specific Enterprise Systems Development

5.4.1. OPM

Object-Process Methodology (OPM) was introduced in 90s by Dori and recently applied to the domain of semantic web for knowledge enrichment. It is used in this domain to unify knowledge representations of both human and machine. OPM is a single-model approach that utilises only one kind of diagram, called Object-Process Diagram (OPD), for modelling the structure, behaviour and function of a system. One of the strength of OPM that it aims at avoiding the consistency and multiplicity problems that might occur when the number of models, used for describing the same system, is increased [7].

The OPD diagram is a visual graph-like workflow diagram, corresponding to a dual-purpose and natural-like Object-Process Language (OPL) script, a subset of natural English, that is based on context-free grammar [31]. Any OPD diagram is automatically translated into a set of OPL. The metamodel of OPD describes two generic concepts, things for expressing objects or processes, and links for expressing various kinds of connections between things.

Specifications of systems designed using OPM are presented in both OPD/OPL [7]. Both representation of OPM modelling language (OPD and OPL) are supported by a CASE tool (OPCAT) for enabling the generation of complete systems from its specifications [32].

5.4.2. DOMMLite

DOMMLite is a declarative domain-specific modelling language (DSL) for defining static structure of database-oriented applications and generating executable code for the basic CRUDS operations. Its concepts are built on other ER-like concepts, such as Eclipse ECore and UML Class diagram, such as Operation, DataType, Classifier, Entity and Property. Moreover, OO principles, such as inheritance, are also adopted in

DOMMLite language [33].

The abstract syntax of DOMMLite is defined graphically by a metamodel, via MOF-based metamodeling approach. The notation of UML Class Diagram is used for representing concepts at the metamodel level. Along with the definition of the DOMMLite metamodel, the textual concrete syntax is formalised using attribute grammars, and implemented using xText, an EBNF-like language for specifying a language syntax.

DOMMLite has a textual notation for specifying its concreate syntax, whereas the execution semantics is defined using several code generators. Both abstract and concreate syntax are implemented using openArchitectureWare (oAW) and Eclipse Modelling Framework (EMF). The development framework of DOMMLite is supported by an GMF-based graphical editor for enabling model management and code generation [33].

5.4.3. Sculptor

Sculptor is a simple model-driven software development framework that uses a high-level textual DSL for generating real enterprise systems with basic CRUD-services. A complete application can be derived from a single model. However, the behaviour logic is always added manually, by developers, as it is not defined in the DSL. Applications in different well-known frameworks, such as Hibernate, Spring and J2EE are well-known can be generated from Sculptor [5].

Sculptor employs domain concepts, such as Entity, Repository, Service and Module, in its textual DSL to be used for programming the target system. The Entity concept, for instance, is used for expressing domain objects in the DSL code. Each entity has a value, a type, attribute(s) and an identity. In the generated system, entities are implemented as Enterprise Java Beans (EJBs) or simply Plain Java Old Objects (POJOs) .

Similar to other textual DSLs, The modelling language in Sculptor is defined using grammar-based approach. The XText and XPand technology are used for implementing and parsing the DSL in Sculptor; and the framework is supported by development environment, including a text editor and a DSL parser under Eclipse [5].

5.4.4. UxiXML

UsiXML is an XML-based markup language that specifies user interfaces (UIs) for various contexts of use, such as Graphical User Interfaces (GUIs) and Auditory User Interfaces (AUIs). It especially intended for context sensitive UIs. The language consists of a textual declarative notation, User Interface Description Language (UIDL), that capturing the detailed specification UI elements, such as widgets, controls and containers, in a platform-independent way. The UsiXML language can be integrated into several formats for describing UIs of web technology, such as cHTML, WML, HTML, XHTML, VoiceXML, Java, and C++ [6].

Different system aspects can be described via many language views, such as Task model, Concept model and Transient model. These views are expressed at multiple levels of abstractions. The Task model, for instance, is a top-level model that describes interactive business tasks and their associated objects. Furthermore, the Transient model is an intermediate model required only momentarily during the development phases of the Final User Interface (FUI) [34].

The abstract syntax of these models is defined by the MOF metamodeling strategy, and expressed using UML Class Diagram. The metamodel of the language is context-sensitivity, that consists of a number of UsiXML Ontological models, namely, Domain model, Context model and adaptation model [6], [34]. The Domain model is used for specifying concepts and user tasks of the domain. The Context model is a compositional model that is utilised for characterizing the context of use in terms of user, platform, and environment. On the other hand, the Adaptation model shows how a UI can be adapted after a change of the context of use. The concreate syntax has both XML-based textual notation and a graphical notation for all kinds of UsiXML models [34].

6. Framework for Modelling Languages Classification

In this section, a framework for modelling languages classification is proposed to instruct developers, with limited software modelling skills, that work with models on the area of Information System Engineering. A domain analysis to existing languages used for developing information systems is applied. Different types of information systems are covered in our domain analysis, such as web, cloud-based and mobile application.

A feature modelling technique is adopted to classify and compare different design choices for modelling languages. Main characteristics from various modelling languages that share common similarity are grouped together into fewer categories. At the end, the resulting framework is documented using a multiple-level feature diagram, containing examples of the modelling languages, for each category, rather than their detailed specifications. It is worth mentioning that the idea of adopting feature diagrams for illustrating a classification framework is discussed in earlier work provided by [18]. The following figure (Fig. 4) demonstrates the major variation features of modelling languages, forming the top-level feature diagram.

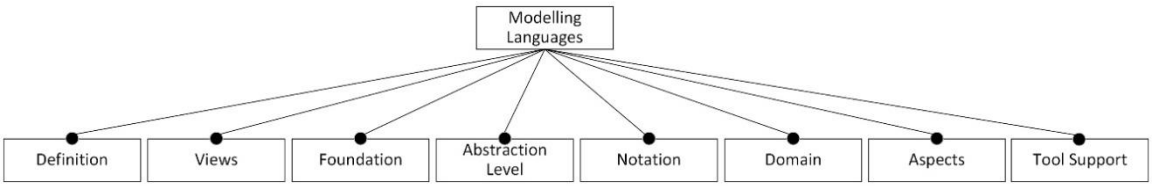

Fig. 5. The feature diagram of modelling languages.

The following subsections discussed each sub-node presented in the Fig. 4 feature diagram. Each section provides a clear meaning of the feature and a brief description regarding its importance.

6.1.1. Views

This feature classifies modelling languages based upon the number of system views used for describing system aspects. It mainly distinguishes languages, that describe the all system aspects in one view, from languages that use different views to express the same system. This feature is considered critical and it taken into account in other comparing and evaluating approaches.

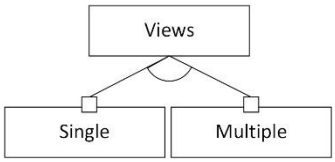

Fig. 6. The Features of views

Some modelling languages support representing the same system in multiple views. Each view represents a particular aspect of that system. Structural and behavioural views are the most common examples of system views in the field of software engineering. One of the main concern of the multiple-view approach is enforcing and maintaining the consistency across the different views [35]-[37]. Figure 5, above, demonstrates the sub-categories of this feature.

On the other hand, there is another modelling language approach that support expressing the system in a single view. A balanced modelling of system structure and behaviour is applied for each model belongs to this approach. This aims at increasing consistency and integrity of the generated code from single-view model [37],[38]. Table 1 compares the two types of modelling languages approaches about this feature.

Table 1. The Comparison of the Various DSMLs with Regard to the Features of Views

	WebML	UML-RSDS	UWE	OPM	WebDSL	WAMM UML	DOMMLite	MobiDSL	MobiCloud	UsiXML	XIS-mobile	Sculptor	OOWS
Single view				✓	✓	✓	✓					✓	
Multiple views	✓	✓	✓					✓	✓	✓	✓		✓

6.1.2. Domain

Some modelling languages are considered domain independent in which it can be used for depicting generic concepts of any type of domain. It provides large sets of constructs and notations used for specifying and documenting various types of software systems. These kinds of modelling language are called General Purpose Modelling Languages (GPML) [33][39][40]. Figure 6 demonstrates the sub-categories of this feature.

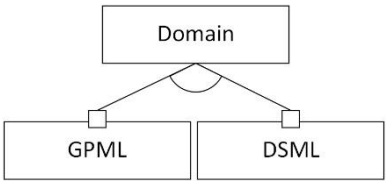

Fig. 7. The features of domain.

On the other hand, there is another type of modelling language that are designed to be suitable only for one specific domain, called Domain-Specific Modelling Language (DSML). These DSMLs contain semantically rich domain concepts, with better power of expression than GPMLs, to allow representing application related concepts in a simple and explicit way [33], [39]. DSMLs use few constructs or concepts that are closer to its application domain. DSMLs are normally easier to read, understand by domain experts [40], [41].

This category is critical when the issue of increasing productivity is considered. It is argued that the GPMLs have not increased productivity and have not given any business benefit when applied to a complete system [42], [43], as the detailed models from which code can be generated are on the same level of abstraction as the programming languages that are supported [44], [45]. It is worth mentioning that the productivity, reliability, maintainability and portability might be increased when generating error-free few lines of code from high level models that are easier to maintain [46], [47]. Table 2 compares the two types of modelling languages approaches about the Domain feature.

Table 2. The Comparison of the Various DSMLs with Regard to the Features of Domain

	WebML	UML-RSDS	UWE	OPM	WebDSL	WAMM UML	DOMMLite	MobiDSL	MobiCloud	UsiXML	XIS-mobile	Sculptor	OOWS
GPML													
DSML	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

From that, the Domain-Specific Modelling Languages can be classified further into several sub-categories based on the targeted domain of interest. To exemplify this, an extended feature level is introduced, Figure 8, with two sub-categories to distinguish between two major DSML kinds, DSMLs for (software) system

modelling and DSMLs for Hardware Design. The DSMLs for modelling software systems is also decomposed into several sub-types to cover all kind of the real-world software systems. In Figure 9, two sub categories are considered for simplicity, as this paper focus more on DSMLs for information systems modelling.

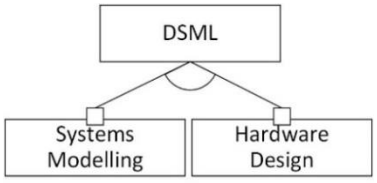

Fig. 8. DSML Sub-categories.

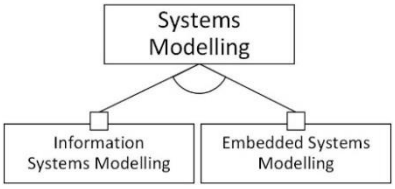

Fig. 9. Software systems modelling.

In the same context, the information systems modelling sub-category can be expanded further to cover various kinds of information system available nowadays (Figure 9). This contributes in the application of the proposed framework to classify modelling languages based on the type of information system they. Table 3, summarises this comparison between the DSLs covered in the survey.

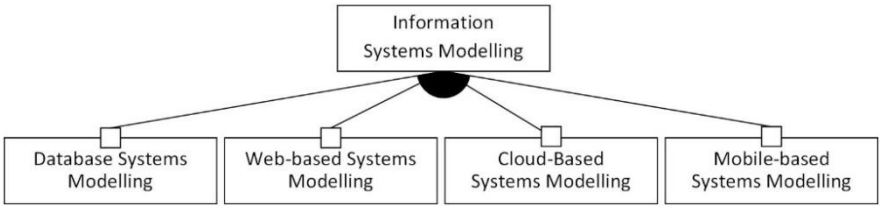

Fig. 9. Sub-categories of information system modelling feature.

Table 3: The Comparison of the Various DSMLs Based on the Type of Targeted Information System

	WebML	UML-RSDS	UWE	OPM	WebDSL	WAMM UML	DOMMLite	MobiDSL	MobiCloud	UsiXML	XIS-mobile	Sculptor	OOWS
Web-based	✓	✓	✓		✓	✓							✓
Cloud-based									✓				
Mobile-based								✓	✓		✓		
Tier-Specific				✓			✓			✓		✓	

6.1.3. Definition

This feature categorises modelling languages based on the way of defining language concepts (abstract and concreate syntax) on the meta-layer. There exist different approaches to the definition of the syntax and semantics of a modelling language, illustrated in Figure 10 [47]. For instance, grammar-based approach (e.g. with Extended Backus-Naur Form; EBNF) can be used to describe a language's syntax in a formal and precise way. In contrast, metamodeling approach, e.g. MOF-based and UML-based (Figure 11), provides a semi-formal approach for defining modelling languages concepts, especially for visual languages that have graph-like structures.

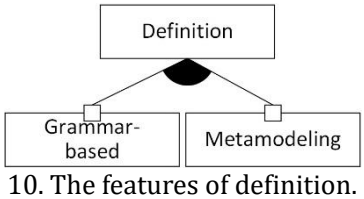

Fig. 10. The features of definition.

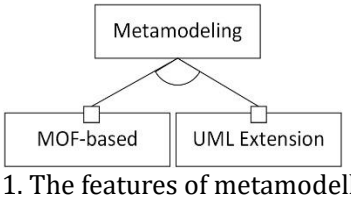

Fig. 11. The features of metamodelling.

The metamodeling approach is classified into several types based on the metamodeling language for defining and realising models of a modelling language. One well-defined metamodeling language used in this context is the UML, via the metamodeling infrastructure of the UML [48]. This infrastructure offers two UML-based extension mechanisms: the lightweight extension mechanism, via UML profiles, and the heavyweight extension mechanism, via extension or modification of the core UML metamodel (Fig. 12) [49], [50].

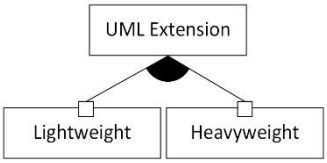

Fig. 12. The features of UML extension.

The UML profile consists of a set of stereotypes that specify UML elements in that profile for the domain of interest. The stereotyped elements have the same structure of traditional UML element plus extra constraints and tagged values that are added, by the stereotype, to that metamodel element [48][49]. Without any conflict in semantics, the profile only customises the metamodel without adding new metaclass in the UML metaclass hierarchy [50].

On the other hand, the heavyweight extension mechanism aims at extending the UML metamodel by explicitly adding new or modifying its metaclasses or associations between these metaclasses [48]. This strategy suffers from a lack of flexibility, which makes the specification of constraints for consistency and integrity checking more difficult than required. It also has weaker CASE tool support in contrast to the UML profiling mechanism [51].

Besides the UML-based approach, defining an entirely new modelling language from scratch means of MOF is a second widely-used approach in this area. It enables adopting the new created metamodel to our preferences and demands. This approach may introduce an entirely new concrete syntax and new CASE tools to support the presentation of DSML elements. It is worth mentioning that building tools that fully support model management is a complex, extensive task requires high effort [51]. Table 4 compares the four major strategies for defining modelling languages.

Table 4. The Comparison of the Various DSMLs with Regard to the Features of Definition

	WebML	UML-RSDS	UWE	OPM	WebDSL	WAMM UML	DOMMLite	MobiDSL	MobiCloud	UsiXML	XIS-mobile	Sculptor	OOWS
Grammar				✓	✓		✓		✓			✓	
MOF	✓						✓	✓		✓		✓	
UML profile		✓	✓			✓					✓		✓
UML Extension			✓										

6.1.4. Foundation

The Foundation feature classifies modelling languages based on the type of the formalisation approach used for formalising the language semantics. The necessity of having formal foundation of modelling languages has emerged as one of the most critical features in model-based verification, validation and simulation task [52]. Modelling languages that are defined, via MOF for instance, can not strictly represent its structural and behavioural semantics. Thus, many significant properties of the language, such as consistency and correctness cannot be systematically validated or verified [53].

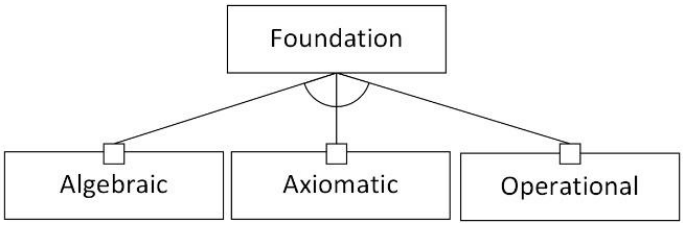

Fig. 13. The Features of Foundation

According to [22], there are many approaches for assigning semantics to a language formally, illustrated in Figure 13. Each approach has its pros and cons and support different forms of reasoning. For instance, the Algebraic approach maps language constructs into mathematical algebra. A further example is the Axiomatic approach that maps language constructs into logical theories using mathematical structures, such as set theory, with axioms for defining the language properties. For additional example, the Operational approach maps language constructs into structures of an abstract execution environment, such as state machine or graph. Table 5 compares the various types of modelling languages based on their formalisation approaches.

Table 5. The comparison of the various DSMLs with regard to the features of Foundation

		WebML	UML-RSDS	UWE	OPM	WebDSL	WAMM UML	DOMMLite	MobiDSL	MobiCloud	UsiXML	XIS-mobile	Sculptor	OOWS
Grammar	Context-free				✓			✓						
	SFD2					✓								
	BNF	✓								✓				
	XText							✓					✓	
UML		✓	✓	✓			✓					✓	✓	✓
XML									✓		✓			
Algebraic														
Axiomatic	RAL		✓											
	FOPL													
Operational														

6.1.5. Notation

Domain-Specific Languages (DSLs), nowadays, are mostly based on textual or graphical representation or on the combination of those two (hybrid) [33], [54], [55]. The Notation feature, in the proposed classification framework, categorises modelling languages into these two main types (Figure 14). In the Graphical modelling languages, developers use a diagram technique with named symbols that represent concepts and lines that connect the symbols and represent relationships and various other notation to represent constraints. On the other hand, they use standardized keywords accompanied by parameters or natural language terms and phrases to make computer executable expressions.

According to [33], [56], there is no definite answer whether textual or graphical notation is better. As there is a widespread belief that graphical modelling languages are easier to understand than those textual ones. However, as it empirically proved, these graphical languages might be worse than textual languages in the comprehensibility of some language constructs.

To exemplify this arguable issue and relate it to tools support, two types of notation editors are observed. Any plain-text editor (e.g. Emacs for Linux, TextMate for Mac OS, and Notepad for Windows) can be used for visualising and editing models based on any textual notations [33]. This is considered advantage that

supports the textual based modelling languages. On the other hand, the graphical notation requires special kinds of visual editor for visualising and editing graphical models, such as the UML-based modelling tools Creately and Lucidchart for both Mac OS and Windows, and Edraw for Linux. Table 6 compares the two types of modelling languages based on their notation types.

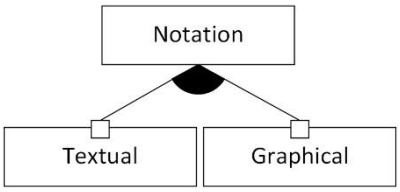

Fig. 14. The features of notation.

Table 6. The comparison of the various DSMLs with regard to the features of Notation

	WebML	UML-RSDS	UWE	OPM	WebDSL	WAMM UML	DOMMLite	MobiDSL	MobiCloud	UsiXML	XIS-mobile	Sculptor	OOWS
Graphical	✓	✓	✓	✓		✓			✓	✓	✓		✓
Textual		✓		✓	✓		✓	✓	✓	✓		✓	

According to [56], the most well-known technique to categorise textual DSLs is related to the way we build its domain-specific semantics. This feature categorises textual modelling languages based upon the way that a language is implemented. The language can be implemented, as an embedding, on the top of an existing host language. This type of DSLs is called internal (embedded). Therefore, the internal DSL gains all benefits, immediately, of the existing programming language (host language), such as grammar, parsers, tools (editors) and compilers. However, the challenge with an embedded DSL is to tactfully design the language so that the syntax is within the confines of what the host language allows, while remaining expressive and concise. Fig. 15 demonstrates the sub-categories of this feature.

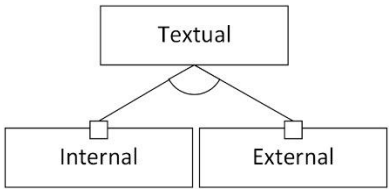

Fig. 15. The Features of textual notation.

On the other hand, the language can be implemented ground-up as an independent new language with its own separated compilation infrastructure, including lexical analysers, parsers, interpreters and code generators. This type of languages is called external (standalone) DSL. It worth mentioning that, the complexity of building a new DSL from scratch depends on how rich concepts the language contains. Table 7 compares the textual modelling languages based on their implementation ways.

Table 7. The Comparison of the Various DSMLS with Regard to the Features of Textual Notation

	WebML	UML-RSDS	UWE	OPM	WebDSL	WAMM UML	DOMMLite	MobiDSL	MobiCloud	UsiXML	XIS-mobile	Sculptor	OOWS

Internal DSL	-	✓	-			-			✓		-		-
External DSL				✓	✓		✓	✓		✓		✓	

6.1.6. Abstraction level

Raising the level of abstraction is another key principle of software engineering. It is considered one of the major point of MDE that tries to achieve to improve developers' productivity. In MDE approaches, a software system is represented in a platform-independent way via several abstracted models. Each platform-independent model expresses a view of the system in a certain level of abstraction.

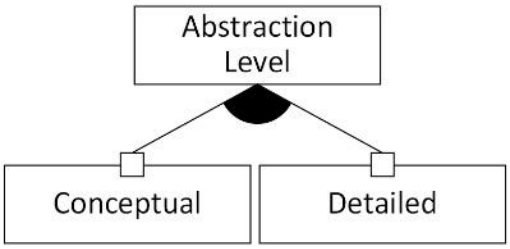

Fig. 16. The features of abstraction level.

In the proposed framework, two generic abstraction levels are considered, namely, Conceptual level and Detailed level (Figure 16 above). The conceptual level represents a logical view of the system without any physical detail, whereas the detailed level includes the physical description of the system as well as the final code. Models at the Conceptual level are regraded high-level models or intermediate models. The high-level of abstraction models hide technical detail and use terminology close to the problem domain, rather than solution domain. It is also known as requirements models [57]. A clear example of this kind of models, is those models used at requirement elicitation and specification phase during the development lifecycle (e.g. the UML Use case diagram).

Besides this, conceptual models such as data flow or UML Activity diagrams are considered intermediate representation of a system. It is also known as analysis models. This kind of models still abstracted and platform-independent of any technical domain, focusing on extra detail of a view of the system. On the other hand, models at the Detailed level are considered low-level model or direct code, which contain richer technical detail for a specific technology (platform-specific). This kind of models are used more during the design phase of development. These models provide an adequate level to enable a complete code generation, such as the UML Sequence diagram. Table 8 compares the modelling languages based on their level of abstractions.

Table 8. The Comparison of the Various DSMLs with Regard to the Features of Abstraction Level

		WebML	UML-RSDS	UWE	OPM	WebDSL	WAMM UML	DOMMLite	MobiDSL	MobiCloud	UsiXML	XIS-mobile	Sculptor	OOWS
Conceptual	Requirements	✓	✓	✓	✓		✓				✓	✓		✓
	Analysis	✓	✓	✓	✓		✓				✓	✓		✓
Detailed	Design	✓	✓	✓	✓	✓	✓		✓	✓		✓		✓
	Code					✓		✓	✓	✓	✓		✓	

6.1.7. Development environment

This feature categorises modelling languages based on the development environment used for creating a

CASE tool or a graphical/textual editor that support model management throughout the MDE development lifecycle of the language. Table 9 compares the modelling languages based on their supporting development environments.

Table 9. The Comparison of the Various DSMLs with Regard to the Features of Development Environment

		WebML	UML-RSDS	UWE	OPM	WebDSL	WAMM UML	DOMMLite	MobiDSL	MobiCloud	UsiXML	XIS-mobile	Sculptor	OOWS
OpenUWE				✓										
WebRatio		✓												
ArgoUWE				✓										
UWEXML				✓										
OPCAT					✓									
Enterprise Architecture												✓		
Eclipse	EMF					✓	✓		✓		✓		✓	✓
	XPand							✓					✓	
Ruby										✓				
UML-RSDS			✓											

6.1.8. Aspects

This feature classifies modelling languages based on the system aspect it represents. The separation of concerns principle is considered one of the crucial principles in software engineering, especially in Aspect Oriented Software Development (AOSD). The AOSD approach supports the decomposition of software designs that align with both requirements specifications and with the final code. It tackles some code problems such as scattering, tangling, and coupling which weaken the reuse capability [58][59]. Figure 17 demonstrates the sub-categories of this feature.

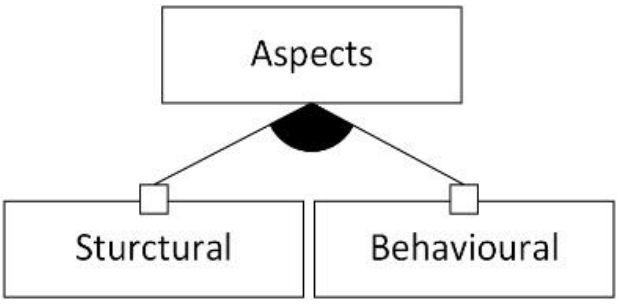

Fig. 17. The features of aspects.

According to [59] and [60], Separating concerns of interest, clearly, resulted in better traceability from requirements to implementation, and effective reuse and evolution, leading to better change management. It is worth mentioning that adopting AOSD technique alongside with MDE enables representing each system aspect via a separate model (a system view), such as, data flow view, security view, protocol view, and database view. This enhances the software modelling activity during the development lifecycle. The following table (Table 10) demonstrates a detailed comparison between DSLs covered in this report with respect to their supported views. Table 10 compares the modelling languages based on the aspects covered.

Table 10. Detailed Comparison of the Various DSMLs with Regard to the Features of Aspects

		WebML	UML-RSDS	UWE	OPM	WebDSL	WAMM UML	DOMMLite	MobiDSL	MobiCloud	UsiXML	XIS-mobile	Sculptor	OOWS
Domain Entities (Structural)	Data	✓		✓		✓			✓				✓	
	Multi-Data	✓												
	Entity						✓	✓				✓		✓
	Object				✓					✓				✓
	Information													✓
	Class		✓	✓										✓
	Repository													
	Concept										✓		✓	
Control Flow (Behavioural)	Process	✓		✓	✓									
	Activity		✓	✓									✓	
	Workflow					✓								
	Logic								✓					
	Controller									✓				
	State machine		✓	✓										✓
	Data flow													✓
	Service	✓												
Presentation (Structural & Behavioural)	Presentation	✓		✓					✓				✓	
	GUI											✓		
	Client HTML		✓											
	Page					✓			✓					
	Screen													✓
	form		✓				✓							
	View		✓							✓				
	Hypertext	✓												
	Navigation			✓			✓		✓			✓	✓	
	User interaction											✓		✓
Architecture (Structural)	Domain										✓			
	Architectural											✓		
	Module													
Business Activities (Structural)	Use case		✓	✓								✓		
	Task										✓			✓
	Domain										✓			
Specific Aspects (Structural)	Access Control													
	OCL		✓											
	Principle (user)					✓							✓	
	Context										✓			
	Adaptation			✓							✓			
	Transient										✓			

7. Discussion

Although there are satisfactory solutions for generating various types of web applications, via MDE approaches, modelling languages adopted in these approaches are still in the stage of investigation and evolving. The aim of the proposed work is to classify various types of modelling language to be able to compare them. Comments in this section are based only on the survey conducted on different modelling languages and published examples of their approaches. In order to achieve the total validity of the proposed

classification system, further experiments on each approach must be conducted.

7.1. General Discussion

According to the analysis results, based on the number of DSL views, illustrated in Table 1, the number of multiple-view modelling languages outweighs the number of the single ones. 61.5% of the modelling languages, covered by the survey, represents an information system via multiple views (models). Furthermore, it is found that the majority of graphical modelling languages supports representing web systems at multiple levels of abstractions (Table 8), with 84.6%. On the other hand, the textual languages are mostly used at low-level design and/or code.

It also can be noticed, from Table 8, that web engineering phases, requirements, analysis, design and implementation, are systematically separated via representing each type of models/code at a layered that it belongs to. This strategy is not fully supported in textual modelling languages approaches. These approaches deal more with the design and implementation (coding) phases. From that, it can be attained that graphical modelling languages may bring further benefits to business users, with less technical and design skills, by raising the abstraction level to hide technical detail that must be considered.

Regarding the way of defining DSLs (Table 4), there is a remarkable popularity in adopting either grammar-based, MOF-based or UML profiles, with 38.5%, for describing metamodel abstract syntax of the languages covered in this survey. Besides this, UML and various types of grammars are common strategies utilised for formalising the semantics of modelling languages covered in the survey (Table 5), in which each approach scores 53.8%.

The amount of support is also based on other factors, such as the complexity of the language and the ease of the development environment. Table 9, for instance, shows that the Eclipse development environment gains a popularity in constructing support MDE tools and modelling editors for both textual and graphical DSLs. In contrast to other environments, Eclipse, through EMF, XText and XPand, is adopted by 61.5% of modelling languages covered by this survey.

7.2. Deep Concept Discussion

According to the analysis results summarised in Table 10, two of the major components appear in all types of information systems are the structure of domain entities and interrelationships between them. This concept is recorded, with different terms, in all modelling language used for developing web information systems. The information modelling terms Data and Entity are adopted by both textual and graphical DSMLs, and appear in 38.5% and 30.8% respectively of the languages covered in the survey. On the other hand, the Object-Oriented terms Class and Object appear, both with 80%, in the UML-based or UML-like languages, such as UML-RSDS and UWE.

Besides. some approaches distinguish, explicitly, between singleton and collection domain objects by using two different terms, such as Data and Multi-Data in WebML, and Entity and Repository in Sculptor. Other languages adopt UML or ER style in expressing cardinality (multiplicity), such as OPM, UWE and UML-RSDS.

The structure and workflow of business (domain) activates, in addition, are other key concepts considered when developing information systems. The activities structure supports the construction of an overall architectural view of the business functions. This concept appears in a separate system model in 38.5% of the modelling languages discussed in this paper. The concept of use cases is used in 23.1% of the DSL presented in this paper, such as XIS-mobile and UML-RSDS, to describe the overall structure of business activities, including aggregation and generalisation relationships. It worth mentioning that few modelling languages consider the external architectural view of the web system and its major components, with 15.4% in all forms.

Furthermore, business workflow (logic) appears as a separate concept or model in the majority of graphical modelling languages covered in the survey, with 69.2%. Some languages have two different concepts/models to represent business logic, such as WebML, UML-RSDS and UWE. Like the structure of activities, with 30.8% for each model, Activity and State machine diagrams appear in many languages for modelling business workflow, such as UML-RSDS, UWE and OOWS. The Process diagram also adopted, with 23.1%, in other DSLs, such as WebML and OPM. However, textual modelling languages either have their business logic directly manipulated in its concrete syntax (code), or target basic CRUDS web applications only, such as DOMMLite Sculptor and MobiCloud.

In addition, information system presentation concepts, including presentation structure and navigation are other key concepts considered core elements, or models, in various types DSMLs for web engineering. Various types of presentation structures, such as Page, View, Form, and Screen are provided commonly, with 76.9%, as main concepts of the DSLs covered by survey. On the other hand, different forms of navigations, including hypertext connection and user interaction appear with 53.8% in various occasions.

During the comparison, it is found that specific aspects gain explicit support by some modelling languages. This strategy encouraged developer to express domain-specific features and constraints aside from any other common concepts, such as security and configuration. For instance, DSLs like WebDSL and OOWS describe user characteristics in a separate language model (view). In the same context, UWE and UsiXML have a special kind of model for representing the adaptivity aspect of web systems.

8. Recommendation

This paper studied a variety of modelling languages, coming from different information system development approaches, and with varying characteristics. The proposed features comparison of thirteen web DSMLs shows significant overlapping between some modelling languages to cover common information system aspects. At the same time, variations and additional domain-specific features are emerged to show substantial differences between some DSMLs when targeting different platform technologies and/or system architectures.

Adopting our classification framework is a useful technique for gathering various complementary information system aspects, taken from different modelling languages, in one user-friendly modelling language for information system engineering.

In the domain of information system engineering, guidelines for designing a user-friendly modelling language, with simpler notation and cleaner semantics than other existing UML-based and DSMLs approaches, are presented in this section. To achieve the ultimate benefits of the proposed classification framework, the resulting guidelines and recommendations are adopted and followed in the detailed design of our promising lightweight DSML. This is a further work that is considered out of the scope of this paper.

Domain. A MOF-based Domain-Specific Modelling Language (DSML) will be considered as a design decision for our promising modelling language for information systems development. Common aspects appear in different types of web systems will be expressed in a common semantic metamodel. Business users will use these high-level generic concepts for constructing their desired information system. Based on the target platform, model transformers, translators and code generators will be responsible for producing all technical detail, design specifications as well as the final executable code automatically instead of the casual designer (end-user) [61], [62].

Views. The Multi-view approach will be considered in the language design, as we presume this suits more the conceptual (natural) thinking of business users. Each critical aspect of the system will be captured in an independent model, using high level domain concepts. These aspect models (views) are interrelated which, together, lead to significant design choices of the final system. We think that it would be easier to raise the

abstraction level for each aspect model independently and rely on sophisticated model transformations mechanism to infer and produce automatically lower-level design detail during the MDD lifecycle.

Notation. A widely-known and adopted graphical notation will be used to represent the concrete syntax of the language visually. We believe this makes the notation realisable and understandable, in a more efficient degree, by business end-user who have limited technical skills. Therefore, the emphasis of our promising approach will be introducing a UML-like or BPMN-like DSML or a combination of both notation that are simplified and semantically cleaned. Moreover, XML will be used as an underlying textual notation to improve interoperability to facilitate exchanging and reusing information internally as well as externally during MDE model transformations and code generation stage [62], [63].

Foundation. The concepts and the notation of the language will be formalised using the Axiomatic approach, via First-Order Predicate Logic (FOPL) with extensions needed. As FOPL is a widely-adopted approach for formalising the constructs of UML diagrams and other DSLs [64]. It mainly used for describing the language semantics and the interrelationships between its elements via mapping the concepts to some FOPL predicates and symbols.

Implementation and tool support. Eclipse platform, via its EMF, and GMF modelling framework, will be adopted as the development environment and the modelling tool that provides a full support of implementing and manipulating our promising DSML models visually and textually. The GMF project, for instance, is capable to generate a fully functional graphical editor in Eclipse, in simple steps.

9. Conclusion

Modelling is a crucial activity in model-driven engineering (MDE) process, because it allows for better understanding of the problem domain, the external software system architecture and the internal software design. To achieve efficient modelling practice that narrows the communication gap between the development team and the system end-users in a standard and formal way, the most appropriate modelling language must be adopted. There is a rapid increase in the number of modelling languages nowadays. These languages have common characteristics shared between them. However, significant variations that distinguish these languages also exist.

A feature model for classification modelling languages is introduced as a contribution of this paper for supporting developers to select a suitable modelling language that serves their needs. The presented classification framework is applied to 13 existing modelling languages for information systems engineering. Three types of information system applications are considered in the conducted survey, namely, mobile application, n-tier web application and cloud-based web application.

Although there are satisfactory solutions for generating basic web applications from conceptual and detailed system models, this is not the case when considering business users, with less technical knowledge, as casual developers. Many MDE approaches utilised rich modelling languages, with a degree of technical detail, for creating and managing design models during the development lifecycle. Other approaches have limited flexibility to produce a specific type of information system that might clash with the actual demand of users. The proposed classification framework can be adopted by casual developers to provide a guidance for they need of choosing a proper language that meets their modelling skills.

References

[1] Brambilla, M., Comai, S., Fraternali, P., & Matera, M. (2008). Designing web applications with WebML and WebRatio. *In Web Engineering: Modelling and Implementing Web Applications*, 221-261.

[2] Groenewegen, D. M., Hemel, Z., Kats, L. C., & Visser, E. (2008, October). Webdsl: A domain-specific language for dynamic web applications. *Proceedings of the Companion to the 23rd ACM SIGPLAN*

Conference on Object-Oriented Programming Systems Languages and Applications.

[3] Ranabahu, A. H., Maximilien, E. M., Sheth, A. P., & Thirunarayan, K. (2011, October). A domain specific language for enterprise grade cloud-mobile hybrid applications. *Proceedings of the Compilation of the Co-Located Workshops on DSM'11*.

[4] Kejriwal, A., & Bedekar, M. (2009). Mobidsl-a domain specific langauge for mobile web applications: developing applications for mobile platform without web programming, *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modelling*, Orlando, USA.

[5] Nordwall, P. The Server Side.

[6] Vanderdonckt, J., & Calleros, J. M. G. (2009). Usixml, a user interface model and language engineering approach.

[7] Dori, D. (2011). Object-process methodology: A holistic systems paradigm. Springer Science & Business Media.

[8] Seidewitz, E. (2003). What models mean. *IEEE Software, 20(5)*, 26-32.

[9] Alotaibi, Y. (2016). Business process modelling challenges and solutions: A literature review. *Journal of Intelligent Manufacturing*, *27(4)*, 701-723.

[10] Alotaibi, Y., & Liu, F. (2017). Survey of business process management: Challenges and solutions. *Enterprise Information Systems*, 11 (8): 1119-1153.

[11] Morais, F., & Rodrigues da Silva, A. (2015). Assessing the quality of user-interface modeling languages. *Proceedings of the 17th International Conference on Enterprise Information Systems (ICEIS 2015)*.

[12] Alotaibi, Y., & Liu, F. (2013). Average waiting time of customers in a new queue system with different classes. *Business Process Management Journal, 19(1)*, 146–168.

[13] Alotaibi, Y., & Liu, F. (2014). An empirical study of a novel managing customer power model and business performance in the mobile service industry. *Business Process Management Journal, 20 (6)*, 816 – 843

[14] Alotaibi, Y., & Liu, F. (2014). A novel secure business process modelling approach and its impact on business performance. *Information Sciences, (277*), pp. 375–395

[15] Belaunde, M., Casanave, C., DSouza, D., Duddy, K., El Kaim, W., Kennedy, A., Frank, W., Frankel, D., Hauch, R., Hendryx, S, & Hettinger, M. (2003). MDA Guide Version 1.0. 1.

[16] Ribeiro, A., & da Silva, A. R. (2014). Evaluation of XIS-mobile, a domain specific language for mobile application development. *Journal of Software Engineering and Applications, 7(11)*, 906.

[17] Krogstie, J. (2012). Model-based development and evolution of information systems: A quality approach, chapter 5, pages 249–280. Springer-Verlag London.

[18] Czarnecki, K., & Helsen, S. (2006). Feature-based survey of model transformation approaches. *IBM Systems Journal, 45(3)*, 621-645.

[19] Alotaibi, Y. (2017). Graphical business process modelling standards, techniques and languages: A literature review. *International Journal of Business Information Systems (IJBIS), 25(1)*, 18-54.

[20] Roeser, T., & Kern, E., (2015). Surveys in business process management – a literature review, *Business Process Management Journal, 21 (3)*, pp.692-718

[21] Lano, K., & Kolahdouz-Rahimi, S. (2010, October). Specification and verification of model transformations using UML-RSDS. *Proceedings of the In 8th International Conference on Integrated Formal Methods (IFM2010)*.

[22] Lano, K. (2009). A compositional semantics of UML-RSDS. *Software and Systems Modeling, 8(1),* 85-116.

[23] Lano, K. (2014). The UML-RSDS manual. *Technical Report*, Department of Informatics, *King's College London*.

[24] Visser, E. (2008). WebDSL: A case study in domain-specific language engineering. *Generative and*

Transformational Techniques in Software Engineering II, 291-373. Springer Berlin Heidelberg.
[25] Visser, E. (2004). Program transformation with Stratego/XT. Domain-specific program generation.
[26] Cadavid, J. J., Quintero, J. B., Lopez, D. E., Hincapié, J. A., Brogi, A., João, A., & Anaya, R. (2009, April). A domain specific language to generate web applications. *In CIbSE*, 139-144.
[27] Kraus, A., Knapp, A., & Koch, N. (2007). Model-driven generation of web applications in UWE.
[28] Knapp, A., Koch, N., Moser, F., & Zhang, G. (2003, September). ArgoUWE: A CASE tool for web applications.
[29] Pastor, O., Fons, J., & Pelechano, V. (2003, July). OOWS: A method to develop web applications from web-oriented conceptual models. *International Workshop on Web Oriented Software Technology (IWWOST)*. 65-70.
[30] Manjunatha, A. K. (2011). A domain specific language based approach for developing complex cloud computing applications. Doctoral dissertation, Wright State University.
[31] Kabeli, J. & Shoval, P. (2005). Comprehension and quality of analysis specifications: A comparison of foom and opm methodologies. *Information and Software Technology, 47(4)*, 271–290.
[32] OPCAT. Rapid conceptual design for complex systems.
[33] Dejanovic, I., Tumbas, M., Milosavljevic, G., & Perisic, B. (2010). Comparison of textual and visual notations of DOMMLite domain-specific language.
[34] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M., & Trevisan, D. (2004, May). Usixml: A user interface description language for context-sensitive user interfaces. *Proceedings of the ACM AVI'2004 Workshop*.
[35] Bashir, R. S., Lee, S. P., Khan, S. U. R., Chang, V., & Farid, S. (2016). UML models consistency management: Guidelines for software quality manager. *International Journal of Information Management, 36(6)*, 883-899.
[36] Mens, T., Van Der Straeten, R., & Simmonds, J. (2003, October). Maintaining consistency between UML models with description logic tools. *ECOOP Workshop on Object-oriented Reengineering*.
[37] Reinhartz-Berger, I., & Dori, D. (2005, January). OPM vs. UML— Experimenting with comprehension and construction of web application models. *Empirical Software Engineering. 10(1)*, 57-80.
[38] Reinhartz-Berger, I., & Dori, D. (2004). Object-process methodology (OPM) vs. UML-a code generation perspective. *In CAiSE Workshops, (1),* 275-286.
[39] Kirchner, L., & Jung, J. (2007). A framework for the evaluation of meta-modelling tools. *The Electronic Journal Information Systems Evaluation*, *10(1)*, 65-72.
[40] Kosar, T., Oliveira, N., Mernik, M., Pereira, M. J. V., ˇCrepinšek, M., da Cruz, D., & Henriques, P. R. (2010). Comparing general-purpose and domain-specific languages: An empirical study. *In Computer Science and Information Systems*.
[41] Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys, 37*,316–344.
[42] Tolvanen, J. P. (2005). Domain-specific modeling for full code generation. *Methods & Tools, 13(3)*, 14-23.
[43] Dalgamo, M. & Fowler, M. (2008). UML vs domain specific languages, *EE Times*.
[44] Kelly, S. (2007). Domain-specific modeling languages: moving from writing code to generating it.
[45] Kelly, S., & Tolvanen, J. P. (2008). Domain-specific modeling: enabling full code generation. John Wiley & Sons.
[46] Cao, L., Rossi, M., & Ramesh, B. (2009, July/August). Are domain-specific models easier to maintain than UML models?, *IEEE Software, 26*, 19-21.
[47] Tolvanen, J. & Kelly, S. (2005). Defining domain-specific modelling languages to automate product derivation: collected experiences. *Proceedings of the 9th International Conference on Software Product*

Lines (SPLC'05).

[48] Hoisl, B., Sobernig, S., & Strembeck, M. (2014, March). A catalog of reusable design decisions for developing UML/MOF-based domain-specific modelling languages. *Technical Reports / Institute for Information Systems and New Media*, WU Vienna University of Economics and Business, Vienna.

[49] Pérez-Martínez, J. E. (2003). Heavyweight extensions to the UML metamodel to describe the C3 architectural style. *ACM SIGSOFT Software Engineering Notes, 28(3)*, 5-5.

[50] Bouchbout, K., Akoka, J., & Alimazighi, Z. (2012). An MDA-based framework for collaborative business process modelling. *Business Process Management Journal, 18(6)*, 919-948

[51] Weisemöller, I., & Schürr, A. (2007, September). A comparison of standard compliant ways to define domain specific languages. *Proceedings of the International Conference on Model Driven Engineering Languages and Systems*, (pp. 47-58).

[52] Jackson, E., & Sztipanovits, J. (2009). *Softw Syst Model 8.*

[53] Jiang, T., & Wang, X. (2012). Formalizing domain-specific metamodeling language XMML based on first-order logic. *Journal of Software, 7(6)*, 1321-1328.

[54] Engelen, L., & Van, D. B. M. (2010). Integrating textual and graphical modelling languages. *Electronic Notes in Theoretical Computer Science, 253(7)*, 105-120.

[55] Mazanec, M., & Macek, O. (2012, April). On general-purpose textual modelling languages.

[56] Ghosh, D. (2010). *Dsls in action* (1st ed.). Manning Publications Co., Greenwich, CT, USA.

[57] Panayiotou, N., Gayialis, S., Evangelopoulos, N., & Katimertzoglou, P. (2015). A business process modeling-enabled requirements engineering framework for ERP implementation. *Business Process Management Journal, 21(3)*, 628-664

[58] Clarke, S., Harrison, W., Ossher, H., & Tarr, P. (1999). The dimension of separating requirements concerns for the duration of the development lifecycle.

[59] Santos, F., Cappelli, C., Santoro, F., Leite, J., & Batista, T. (2012). Aspect-oriented business process modeling: analyzing open issues. *Business Process Management Journal, 18(6)*, 964-991

[60] Kulkarni, V., & Reddy, S. (2003, September). Separation of concerns in model-driven development. *IEEE Softw, 20(5)*, 64-69.

[61] Subahi, A. F., & Simons, A. J. (2011, September). A multi-level transformation from conceptual data models to database scripts using java agents. *Proceedings of the International Workshop on Composition and Evolution of Model Transformation*, Kings Collage, London.

[62] Subahi, A. F. (2015). A business user model-driven engineering method for developing information systems, Doctoral dissertation, University of Sheffield.

[63] Khan, Y. A., & Mahmood, S. (2016). Generating UML sequence diagrams from use case maps: aa model transformation approach, *Arab Journal of Science and Engineering, 41(965).*

[64] Jiang, T., She, Y., & Wang, X. (2016). An approach for automatically verifying metamodels consistency. *International Journal of Simulation-Systems, Science & Technology, 17(27)*.

[65] Portugal, I., Alencar, P., & Cowan, D. (2016). A survey on domain-specific languages for machine learning in big data.

[66] Alotaibi, Y. & Liu, F. (2013). Business Process Modelling towards Derive and Implement IT Goals. *Proceedings of the 8th IEEE Conference on Industrial Electronics and Applications*.

[67] Alotaibi, Y. & Liu, F. (2012). A new framework to model a secure e-commerce system. *International Journal of Social and Human Sciences*, *6(6)*, 162-168.

[68] Alotaibi, Y. & Liu, F. (2012). How to model a secure information system (IS): A case study. *International Journal of Information and Education Technology*, *2(2)*, 94-102.

Ahmad F. Subahi is an assistant professor and the head of Department (HoD) at Department of Computer Science, University Collage of Al Jamoum, Umm Al-Qura University, Makkah, Saudi Arabia. He received his PhD from the Department of Computer Science, University of Sheffield in 2015. His research is focused on software engineering and computer science. Specific research areas include model-driven engineering, domain-specific modelling languages, code generation and programming languages design.

Youseef Alotaibi is an assistant professor in the Department of Computer Science at University Collage of Al Jamoum, Umm Al-Qura University, Saudi Arabia. He completed his PhD from the Department of Computer Science and Computer Engineering, La Trobe University in, Melbourne – Australia in 2014. He received his master in information technology (computer network) from La Trobe University. His research interests include business process modelling, business process reengineering, information system, security, business and IT alignment, software engineering, QoS and WiMAX.