International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:7, No:7, 2013

Domin-Specific Language for Enabling End-Users Model-Driven Information System Engineering

Ahmad F. Subahi and Anthony J. H. Simons

***Abstract*—**This Paper presents an on-going research in the area of Model-Driven Engineering (MDE). The premise is that UML is too unwieldy to serve as the basis for model-driven engineering. We need a smaller, simpler notation with a cleaner semantics. We propose some ideas for a simpler notation with a clean semantics. The result is known as μML, or the Micro-Modelling Language.

***Keywords*—**Model-driven engineering, model transformations, domain-specific languages, end-user development.

I. Introduction

IN the real-world, there is a demand to adopt an accurate information system that satisfies the requirements and used effectively for the business. However, having vague or misinterpreted requirements causes errors and extra costs. Therefore, domain experts, who clearly understand the business logic, goals and aware of what exactly they need inside organisation without professional software developing skills, should play key roles in the development lifecycle using high level tools. There are many approaches that aim to tackle these issues and reduce the gap between initial requirements and implementation and accelerating the development process, such as Model-Driven Engineering (MDE), Domain-Specific Languages (DSL) and End-user Development (EUD).

Although the Unified Modelling Language, UML [13], is commonly used to express structural and behavioural of a system within MDE approaches, it suffers from semantics ambiguity and complexity issues. This produces inconsistent UML models difficult to interpret [12]. This lack of formality and more make UML cannot serve as a basis for MDE from business-users' perspective.

II. Background

The general strategy of Model-Driven Engineering aims to capture system specifications through the employment of Models that are expressed at a very high level of abstraction, without technical platform-specific details. It supports integration and interoperability, improves software quality and reduces a development cost [1]. These levels of abstraction are designed using two possible ways: the UML Meta-Object Facility (MOF) via "Profiles" using limited and restricted extensibility mechanisms [3], or appropriate DSLs for a particular domain with an execution engine [2].

The UML-based Web Engineering (UWE) approach, for instance, uses UML profiles to construct DSL for building web applications in more flexible way. It is considered a lightweight extension that captures business process, presentation and navigation aspect of the web systems. UML CASE tools, which support UWE, provide a semi-automatic generation of web software and employ different languages for implementing model transformations [14].

Extended UML models such as Use case, Activity, Sequence, State-Transition and Class Diagram are used intensively to construct the structure and behaviour models. Therefore, an adequate degree of modelling skills and UML awareness are necessary in order to build syntactically and semantically valid UWE models.

Alternatively, building a suitable Domain-Specific Language from scratch is another way to define metamodelling architecture. The DSL represents the syntax and semantics of models with simpler details (subset) than UML. The semantics is defined either by code generators or model transformations using the sufficient information from models [4].

WebML, for instance, is a DSL for web engineering. It allows specifying the conceptual model of web applications, such as data, service, navigation and processes. WebML is supported by tool for code generation [10], [11].

Users in WebML specify all composition and navigation features of their web application using a number of designing languages. The modelling process starts with constructing the data model and ends up with designing the hypertext and presentation view. According to that, end-users must act as a web designer to design each part individually of the system.

The End-User Development (EUD) is a development technique that aims to empower end-users, without technical knowledge, to become involved in the process of designing and/or customising their systems to increase their productivity and satisfaction [5]. It also aims translate accurately and comprehensively the informal description of domain problems to reduce the gap between the exact user desires and what functionalities the implemented system has [8].

Producing end-user tools for constructing web applications, such as DEMIN and Mashups, CBEADS [7] is a widely-

Ahmad F. Subahi is with the Umm Al-Qura University, Makkah, Saudi Arabia. He is now a PhD candidate in Computer Science Department, University of Sheffield, Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK (e-mail: AFSubahi1@sheffield.ac.uk).

Anthony J. H. Simons is with the Department of Computer Science, University of Sheffield, Room 119, Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK (e-mail: A.J.Simons@sheffield.ac.uk).

known example of applying EUD in the real-word to tackle the problem of the lack of web developing skills for non-programmers [6]. This kind of tool considers business-users' perspectives or mental model [6], encodes developers' knowledge as rules [9] and enables users to easily tailor software to meet their individual needs [5].

Ginige and De Silva have introduced a metamodelling approach for enabling end-users to be involved in the continuous development process, side by side with developers, with little technical knowledge. It also aims to enable them to employ effectively the metamodels in such as customisable environment [7].

The concepts in the metamodel are similar to those in the UWE approach [14]. However, all common aspects are grouped at a separate level of abstraction for describing information systems, namely, *Shell*, *Application* and *Function* level [8]. These models are embedded into a component-based Shell introduced as developers' templates for end-users to instantiate a metamodel instance and populate it at one or more levels. The CBEADS and other related (SMART) tools are used to generate business objects and, functions, user's interfaces and SQL queries [7].

In overall, we can realise that current MDE tools required skills of dealing complex models that are syntactically and semantically unclear. This prevents end-users to contribute efficiently and express formally their functional requirements they need. This paper discusses our vision to handle this issue.

III. Micro-Modelling Language (μML)

The premise for this paper is that UML is too unwieldy to serve as the basis for model-driven engineering. The models in UML are too complex and eclectic to be given a single, clear interpretation, while paradoxically not covering all of the views that are needed to completely specify a software system. We propose some ideas for a simpler notation, with a cleaner semantics, in which the iconography is more consistent. Individual models are smaller and more restricted; but there are more kinds of model to cover the different interlinking views of a system. As a result, it is possible to specify partial and total transformations between different kinds of model. The result is known as μML, or the Micro-Modelling Language.

μML aims at raising the level of abstraction to suit business end-users, enabling them to construct their system easily, using less technical knowledge in an efficient way than in existing approaches. This tackles some issues in requirements elicitation to accelerate the development process and meet end-user requirements. Furthermore, it reduces the ambiguity of requirements and troubles that occur during client-designer communication.

A. Task Model

The task model is a structural model that describes the breakdown of some business in terms of the goals and tasks it performs, as well as human interaction and other external system participations. It will typically capture the wider context of the business, within which some software system is to be developed. The intention is that this model should replace the use case diagram in UML [13].

The task model supports capturing tasks (ellipses) at different granularities and encourages the designer to explore task decomposition or composition (diamond arrowhead), until a homogeneous view of the business is obtained. In Fig. 1, the Circulation task is decomposed into An Issue Loan and/or Discharge Loan task.

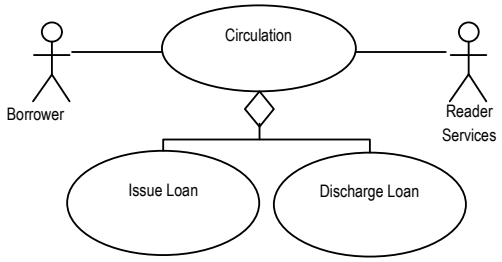

Fig. 1 The task model of a part of a library system

B. Impact Model

The impact model is used to capture the data produced and consumed by tasks in the business. Working out what kinds of data is an intermediate step in requirements capture, which is not well supported in UML [13]. Knowing what data is produced or consumed by tasks constrains the possible orders of task execution.

The impact model captures a partial order on tasks, induced by data dependency. In Fig. 2, a Borrower and a Copy must exist before a Loan involving these objects can be created. Therefore, the tasks that add the Borrower and Copy to the system (not shown) logically precede Issue Loan.

The nodes appearing in the impact model are tasks (ellipses) and logical information, physical objects or people (rectangles). The only arcs appearing in the impact model are the *Create* (solid arrowhead), *Read* (open arrowhead), *Update* (double-ended), and *Delete* (star arrowhead) flows.

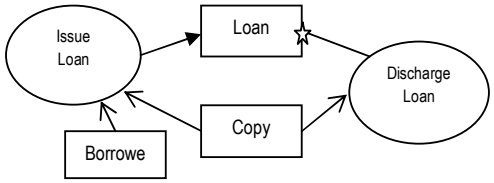

Fig. 2 The impact model of a part of a library system

C. Information Model

Describing data in terms of conceptual entities, attributes and relationships is a standard approach taken in software engineering and is one of the aspects that UML supports in its class diagram [13]. However, the UML class diagram notation mixes up two distinct levels of detail in information modelling: one which is concerned with initial perceptions and the other which is concerned with detailed design.

Here, we view conceptual modelling as a separate activity from database design. The model is common for physical

objects or documents, shown as rectangular nodes, to require further logical decomposition, especially if they contain repeating groups of data. The purpose of an Information model is to capture sufficient information used by the system *ab initio* to generate a Data model.

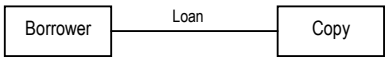

Fig. 3 The information model of a part of a library system

D.Data Model

The data model is intended to represent the logical data, as rectangular nodes, of the system and support development to a point where a logical database schema may be generated. It may be constructed directly from an impact analysis of events (Impact model). Alternatively, the data model may be constructed from a multiplicity analysis of conceptual associations (Information model), as discussed in section 5. Either or both of these prior models may be used as a source for the data model, and may be crosschecked for consistency.

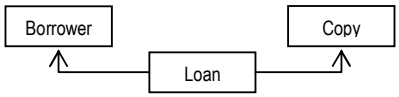

Fig. 4 The data model of a part of a library system

A dependency in the Data model is drawn as a directed edge, with an open arrowhead, to represent dependency of one object upon another. In Fig. 4, a Loan object is dependent upon the Borrower and Copy that it relates.

The data model may also be used to reverse-engineer constraints upon the prior models. If dependency is known, then multiplicity is partially predictable. Read/write and creation/deletion constraints may be inferred likewise.

E. Data Flow Model

The work on the data flow is not completed yet. The aim is to depict the flow between the tasks (ellipses) act on data, and data stores (rec that retrieve the data). The model introduces represents different types of flows, as well as structured text on flows to indicate variables, attributes, and constraints. This level of details and the distinction in the types of flow can lead to derive possible control flows of the tasks and GUIs behaviour that need business-user decisions to resolve complex constructs.

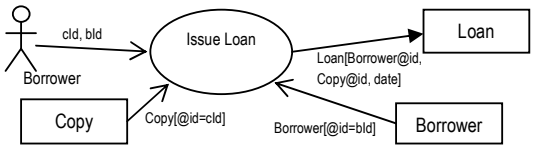

Fig. 5 Data flow model of a part of a library system

IV. Model Transformation

Model transformations are a common task in all MDE approaches and play a key role in mapping models between different levels of abstraction. There is a number of model transformation approaches and languages have emerged in the last few years. They can be viewed as Declarative Languages for performing MDE, such as Atlas Transformation Language (ATL), UML-RSDS, and Kermeta. Here, we are using the (imperative) direct manipulation approach for manipulating model transformations [15].

The starting point is high-level models, close to the business domain, expressed using the proposed μML language. The envisaged processing involves some model-to-model translation steps, mapping a source model to a different target models; and some model-to-code generation steps. How the various high-level models will be combined, possibly "folded" together in the style of aspect-oriented programming, is currently our open research question.

A. Transforming the Impact Model into the Data Model

Tracing the CRUD effects, in the Impact model, of a single task execution on individual objects (viz. a set of individuals) informs the notion of data dependency in the Data model. For instance:

1) A task reading from two objects supposes some kind of association between them, whose properties cannot be further elaborated.
2) A task reading two objects in order to create a third object builds a structure in which the third object depends on the other two.
3) A task reading from two objects and updating the second object implies that the second object depends on the first one.
4) A task reading from one object and destroying a second object implies that the second object depends on the first one.
5) A task reading exclusively from one or other object implies these may be generalised in a disjoint fashion.

In the same context, the rest of mapping rules, including mapping Objects into Entities (collection of objects of the same type), are treated likewise. Fig. 2 shows that a new Loan is created for a pre-existing Borrower and Copy.

Alternatively, the information model is transformed until all the associations are many-to-one (or one-to-one). A many-to-one association taken from the information model is always resolved in the direction from the many to the one in the data model. This is because multiple objects may be created and deleted on the many-side for each object on the one-side. If ever the object on the one-side is deleted, this results in a *cascading deletion* of objects on the many-side. For instance, a many-to-many Loan association between a Borrower and a Copy the information model (Fig. 3) is promoted to an object which depends on its related parts (Fig. 4).

V. Future Work and Conclusion

The current work includes designing other simple and semantically-cleaned models to capture other views of information systems, such as Task Flow model and GUI-State. Transformation challenges, between the current models and the "in-progress" one, are need to be identified and solved. In order to reach this level of clearance, a moderate work is carried on specifying the structure of each model formally using First Order Logic (FOL) with equality. In addition, the rules of transformations will be expressed similarly.

In overall, we have discussed the main ideas behind our research on the Micro-Modelling Language (μML) approach. Fairly sophisticated rules to generate a detailed Data model from the Impact/Information models have been introduced.

References

[1] Almeida, J.P., et al. (2004). On the Notion of Abstract Platform in MDA Development. In Eighth IEEE International Enterprise Distributed Object Computing Conference. IEEE CS Press.
[2] Kelly, S. and J.P. Tolvanen. (2008). Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Press.
[3] Clark, T., et al. (2004). Transformation Language Design: A Metamodelling Foundation. In Second International Conference. Rome, Italy: Springer Berlin, Heidelberg.
[4] Mannadiar, R. and Vangheluwe, H. (2010). Domain-specific engineering of domain-specific languages.In Proceedings of the 10th Workshop on Domain-Specific Modeling. New York, NY, USA: ACM.
[5] Spahn, M. and Wulf, V. (2009). End-User Development of Enterprise Widgets.In Proceedings of the 2nd International Symposium on End-User Development (IS-EUD '09). Springer-Verlag, Berlin, Heidelberg.
[6] Rode, J. and Rosson, M. and Perez-Quinoneones, M. (2004). End-Users' Mental Models of Concepts Critical to Web Application Development. In Proc of the IEEE Symposium on Visual Languages-Human Centric Computing. Washington, DC, USA: IEEE Computer Society.
[7] Ginige, A. and De Silva, B. (2007). CBEADS©: a framework to support meta-design paradigm. In Proceedings of the 4th international conference on Universal access in human computer interaction: coping with diversity. Springer, Berlin, Heidelberg.
[8] De Silva, B. and Ginige, A. (2007).Meta-model to support end-user development of web based business information systems. In Proceedings of the 7th international conference on Web engineering. Como, Italy, 2007.Springer-, Berlin, Heidelberg.
[9] Pinggera, J. et al. (2010). How the structuring of domain knowledge helps casual process modellers. In Proceedings of the 29th international conference on Conceptual modeling (ER'10).Vancouver, Canada 1-5 November 2010. Springer-Verlag, Berlin, Heidelberg.
[10] Brambilla, M., P. Fraternali, and M. Tisi (2008). A Metamodel Transformation Framework for the Migration of WebML models to MDA. in MDWE, CEUR Workshop. CEUR-WS.org.
[11] Moreno, N., P. Fraternali, and A. Vallecillo. (2007). WebML modelling in UML.IET Software.
[12] Naumenko, A. and Wegmann, A. (2003). Triune Continuum Paradigm and Problems of UML Semantics.
[13] Object Management Group: Unified Modeling Language (OMG UML) Superstructure, Version 2.3, 5 May (2010) [online] Available at: omg.org/spec/UML/2.3
[14] UML-based Web Engineering (2011) [online] Available at: uwe.pst.ifi.lmu.de/aboutUwe.html [accessed 18 October 2012].
[15] Mens, T., Gorp, P. V. (2006). A Taxonomy of Model Transformation. Electronic Notes in Theoretical Computer Science, 152, 125-142.