Pondering on the Key Functionality of Model Driven Development Tools: the Case of Mendix

Martin Henkel, Janis Stirna

Department of Computer and Systems Sciences, Stockholm University
Forum 100, SE-16440, Kista, Sweden
{martinh, js}@dsv.su.se

Abstract. Model Driven Architectures and Model Driven Development (MDD) have been used in information system (IS) development projects for almost a decade. While the methodological support for the MDD process is important, the success of a project taking the model driven approach to development also heavily depends on the tool. The tool simply needs to support a set of key functionalities, such as an appropriate level of model abstraction, the refinement of models and finally the execution of models. In this paper we analyze a new MDD tool, namely Mendix, with respect to a number of functionality areas needed to achieve success in a project and capitalize on the benefits of MDD. Our findings are that Mendix use a well selected set of models and that these models are well integrated and suitable for the construction of small systems. Based on the key functionality areas we also point out the weaknesses of the tool.

Keywords. Model Driven Architecture, Model Driven Development, MDD, CASE tools

1 Introduction

The idea that models should drive software development is more than 20 years old by now. Ambitious modeling methods where developed during the eighties to capture requirements on information systems. During the nineties proprietary CASE (Computer Aided Software Engineering) tools were used to partially generate information systems based on models. In some cases the tools were only able to generate code templates and the rest of the code had to be written in a conventional way. More modern approaches to use models for software development include OMGs Model Driven Architecture (MDA) [OM03] and executable UML (xUML) [Me02]. One common denominator in all these approaches is that the proper use of models in software construction raises the abstraction level [St06] [Kl03] [Ha06]. One of the benefits of model-driven approaches is that developers working on a higher level of abstraction can cope with more complex systems with less effort [Kl03]. Parallels can be drawn to the evolution of programming languages from assembler to modern high-level programming languages such as C# and Java. Some authors point

towards the use of models for software development as the next step in this evolution [La09], [Se03].

Even thought there is a common fundament for model driven development (MDD) approaches, there is still a great variation in both the approaches and the tools used to apply them. Each tool and approach has its own benefits and drawbacks. E.g., OMGs MDA approach focuses on the use of several levels of models and transformation of model between these levels and into code. The generated code can then be elaborated to add details missing in the models. This can be put in contrast to xUML where it could be feasible to generate, and execute 100% of the needed code [Ra04].

Stark contrasts can also be seen when comparing old proprietary CASE tools such as Oracle Forms and Microsoft Access Forms to tools based on open standards and platforms, such as Compuware's MDA and Java based tool OptimalJ. Both of these types of tools can be considered to support MDD, as it is "a software engineering approach consisting of the application of models to raise the level of abstraction" [Ha06]. However in forms-based CASE tools such as Oracle Forms, a drawback is that there is very little control over how the models are transformed into code, while in tool such as OptimalJ the developer can create his/her own model transformation as thus have full control over the generated code. So, even thought the fundament of working on a higher abstraction level is shared among MDD tools, there is still a need to distinguish between the features that the tools do or do not support.

In this paper we examine a new tool for model-driven development, Mendix Business Modeler. Mendix contains an interesting mix of models for describing information need and behavior as well as an integrated tool for user interface design. This mix puts the tool in between traditional form-based CASE tools, tools based on executable models and "fully-blown" MDA tools. To highlight the benefits and drawbacks of the tool we use a set of key functionally areas based on literature in the area of MDD. Furthermore, we briefly compare it to well-known MDD approaches and tools. Our ambition is not to do a comprehensive evaluation of the tool, rather we would like to put it in context to MDD tool functionality and MDD approaches in general. The research approach consists of literature analysis and laboratory experiments with the tool.

The remainder of the paper is organized as follows. In section 2 we provide a list of key functionally areas for MDD tools based on literature. In section 3 the Mendix tool is presented by using a simple example case of a web-based complaint reporting system. In Section 4 we analyze the functionality of the tool, based on the key areas presented in Section 2. Section 5 briefly summarizes our findings in terms of benefits and drawbacks of the tool. Concluding remarks and issues of future work are given in Section 6.

2 Desired Functionality Areas for MDD Tools

The main objective of MDD tools is to automate the transition from information system design to implementation. The underlying assumption in this case is that the design is expressed by models from which executable implementation of the system is produced by a tool without further human intervention.

To a large extent MDD tools should be able to contribute to the same business goals than CASE tools, namely, improving software quality, increasing developer productivity, improving control of the development process, lowering development costs, lowering maintenance costs, reducing development backlogs, as well as increasing customer satisfaction. The fulfillment of several of these goals is not only dependent on the tool in use, but also on the intentions of the organization applying the tool and the situational context in which it is applied. In this paper we will only be able to provide argumentative discussion on the capabilities of Mendix to contribute to these goals. In the reminder of this section we will discuss a set of more specific functionality areas that are relevant for assessing MDD tools. We draw these functionality areas from literature in the field of MDD. To get an overall structure, we divide the functionally areas into modeling support and development support.

Modeling support

Basic support for modeling includes a language, or several integrated languages, for representing the concepts in the problem domain. The support for modeling based on these languages is furthermore influenced by several areas of supporting functionality. The list of functionalities that a MDD should support can be made long, however, in order to highlight the MDD tool under study, Mendix, we select a set of key functionalities. Thus, based on literature we point towards the support for abstraction, understandability, executability and model refinement as key areas for a MDD tool.

- *Abstraction*. [Se03] argues that a model should provide means for abstraction, i.e. hiding details that are irrelevant for a given viewpoint. In an MDD project this is relevant because different developers and stakeholders use model for different purposes. Several authors points toward the use of abstraction as a mean to increase productivity when it comes to software development [Ha06] [St06] [Kl03] [Me02] [La09]. At different development stages there can be a need for the tool to support different levels of abstraction of the system design. E.g. in the case of using the MDA approach, first a Computation Independent Model is developed followed by a Platform Independent Model and subsequently by a Platform Specific Model. Separation of technical platform concerns and the models is also something pointed out by several sources [Ra04][Me02][La09].

- *Understandability*. [Mo03] define understandability as the ease with which the concepts and structures in the data model can be understood. In the absence of code in MDD projects models are used for communication within the development team and with stakeholders. Hence, understandability is one the main factors that determine how successful the project team will be able to communicate and work. [Se03] suggests that models should be expressed in intuitively predictable notation. While in principle, this could be any modeling language, the widespread use of UML, probably suggests that it should be preferred in project that do not have other specific requirements for model representation. [Ma05] also emphasize the need to be able to use all features of the modeling language as opposed to a selected subset of UML.

- *Executability*. [Se03] calls this aspect of a model "predictiveness" and argues that the developers should be able to use the model to predict the systems interesting but not obvious properties either by experimentation or by formal analysis. In an MDD setting this means that a model should be executable even if it is incomplete thus

contributing to incremental development of a design artifact. [Uh08] further points out that the current tools do not really address this challenge adequately.

- *Model transformation and refinement support*. [Ri06] points out that for a mature level of MDD usage there is a need for the support for refinements of models, and the definition of refinements in the form of model transformations between model levels. Refinement goes hand-in-hand with the desire to use abstractions. A tool with refinement support allows the designer to define custom refinement rules/model transformations. Furthermore, MDD tools' main feature that distinguished them from visual programming tools is modeling. Hence we should also consider that a good and efficient modeling environment would also have to support model refinements by improving model quality aspects such as completeness and correctness (see [Mo03], [Kr06]). [Se03] in this context talks about accuracy, which is an aspect of completeness (all facts of the real world are represented in the model) and correctness (all facts in the model are represented according to the syntactic rules of the modeling approach). [Uh08] poses a requirement of representation and editing in a textual form.

Development support

Another aspect of MDD tools is support for the development process. Based on the literature, we define six areas of process support:

- *Observability*. [Se03] puts forward a requirement called "model-level observability" which means that an MDD tool should report errors to the developers in similar way compilers and debuggers do – e.g. indicating the line of code that caused the error. MDD should also be able to point to the model component that caused the error. This is called "model-level debugging" by [Uh08].

- *Collaborative development support*. Since the product is developed by a team, functionality for similarity checking and model merging might also be needed. [Uh08] and [Te09] argue that in real life setting the many tools lack efficient support for comparing and merging model artifacts such as graphical views, forms, dialogs, and property sheets as well as text. [Se03] pushes on the importance of support for large teams.

- *Turn-around-time*. In this respect [Se03] points out the importance of two metrics system, size and compilation time consisting of full system generation time and turn-around time for small incremental changes. The latter also is influencing executability and the overall productivity of the team because developers like frequently checking the result of their actions.

- *Integration*. MDD tools, like all tools used for software development, should fit the development environment of the organization, i.e. they should enable integration with other systems such as a company's ERP systems, and legacy systems.

- *Developer competence support*. One of the main values provided by the MDD approach is that it allows involving in the development a broader range of people who know the business needs of the company. This is possible because not all of these people need to have high level programming skills and in-depth understanding of information system architectures. In this context [Ri08] argues that MDD allows much of the system functionality being developed by, so called, "scripters" which requires involvement of fewer senior developers. On the other hand, study reported in [Te09] suggests that the current lack of modeling experts in organizations is a significant challenge affecting MDD adoption.

- *Reuse*. Many CASE tools neglect reuse, and this tendency has to some extent been visible in MDD tools as well. On the one hand reuse can be supported by a way of working and by conscious efforts of the developers to retrieve potentially reusable chunks from earlier products. In this case we are mostly talking about unstructured and ad hoc reuse, i.e. "salvage reuse", and the tool should mostly be used for browsing, cutting and pasting. On the other hand, controlled and conscious reuse can greatly increase efficiency of software development and ideally, an MDD tool would have to support activities such as definition of reusable components, patterns and best practices, as well as searching, retrieving and tailoring them. In addition, [St06] notes that the reuse of central frameworks is an important part of any MDD tool.

3 The Mendix MDD Tool

The MDD tool that we examine in this paper, Mendix, is build by the Mendix company, the Netherlands. Initially started in 2005, the company has had a steady growth, and now employs about 60 persons. Even though it is still small, the company has been given recognition for its tool, recently the company was listed as "cool vendor of 2009" for its innovations and impact by Gartner [No09]. The focus of the company is to develop and market tools and services needed to quickly build web-based systems. Even thought the company offers services, such as consultancy and on-line hosting of systems built using the Mendix tool (a Plattform-as-a-service, PaaS offer) the focus of this paper is on the Mendix tool.

The Mendix tool consists of four main parts:

- *Mendix business modeler*, a Windows based tool that allows a designer/developer to create models.
- *Mendix model repository* that stores the models.
- *Mendix business server* that acts as a web-server, hosts the database, and executes the models in the model repository. The server is based on open-source Java EE platforms and can be installed on both Unix and Windows.
- *Mendix rich client*, which allows the user interface of the created systems to run in a web browser. At run-time the client communicates with the Mendix Business Server through AJAX calls.

Besides the above mentioned main parts, Mendix also provides support for integration through its connectivity manager and integration interface, as well as mobile web clients are supported through "Mobile Forms".

To build a system using Mendix a designer firstly uses the business modeler to create the necessary models (the types of models used will be described later). As a second step these models are deployed to the model repository. As a third step the system can be started by pointing a web-browser to the address of the business server. When a web page is requested from the Business Server, the server interprets the models and creates the needed HTML and JavaScript that is sent to the client. It is interesting to point out that the Mendix business server interprets the models in order to run the systems. This differentiates the tool from others, such as OptimalJ, that relies heavily on code generation in order to build systems.

The Mendix tool allows building web-based systems by interrelating models of both the system behavior and structure. The following three main model types are used to build a system using the Mendix business modeler:

- A "*meta-model*", an information model depicting the information structure of the system. The syntax used for these models are a Mendix specific graphical notation, however the use of UML class diagrams are being considered for a future release.
- *Form-models*, depicting the system user interface consisting of menus and forms. Forms are drawn using simple graphical user interface widgets, such as tables, input fields and buttons.
- "*Microflows*", process models depicting special procedural logic that needs to be executed. The notation used for these models are similar to UML activity diagrams, with a few extensions.

In order to illustrate how models are used to build a system using Mendix we will use a small case making use of the above models. The target is to create a small web based application that can be used to document complaints on newspaper deliveries to newsstands. By using the application users will be able to create, edit and delete customers and the complaints that they might have on paper deliveries. The system will also contain a small process that determines the prioritization of the complaint handling. In order to build the application we need to follow three steps; 1) creating a "meta-model", 2) creating user interface forms, and 3) creating a microflow to handle the complaint prioritization logic. In the description of the types of models extra attention is paid to the relation between models.

3.1 Creating the "Meta-model"

To define the information structure that the system should handle we use the business modeler to create a "meta-model". Note here that the naming "meta–model" is somewhat inappropriate, because the model is an ordinary information model. To be strict, a Mendix "meta-model" is located on what OMG [OM03] refers to as the "M1" level, rather than on the "M2" level where meta-models usually resides.

For our example application we create two "objects" in the modeler, a Customer object and a Complaint object. The resulting model is shown in Figure 1.

Figure 1, Mendix "meta-model" of the example case

As shown in the figure, Mendix uses its own notation for associations between objects. The association should be interpreted as "A Complaint refers to exactly one Customer", "A Customer may have zero Complaints" and the arrowhead indicates that the association is recorded in the Customer. Readers familiar with UML class

diagrams will note that the reading direction of the multiplicities of the associations and the use of the arrow symbol is totally different when drawing a Mendix "meta-model".

In addition to allowing the specification of objects, their attributes and associations the meta-model also allows the designer to specify how the model should behave at run-time. Validation rules can be set on attributes, stating that they are mandatory, or giving a range of valid inputs. Furthermore "delete behavior" can be set, for example indicating that all complaints belonging to a customer should be deleted when the customer is deleted.

Object to forms and object to microflow relations

The specified objects can be related to the other types of models (forms and microflows) in several ways. It is possible to trigger a microflow by setting "events" on the objects- For example, a microflow can be triggered whenever a Complaint is updated or deleted. Furthermore, the tool supports including possible image representations of an enumerated ("enum") attribute. These image representations can be used in forms to display the attribute. Figure 2 shows how the Prio enumeration attributes of the Complaint object is associated to a set of values and images.

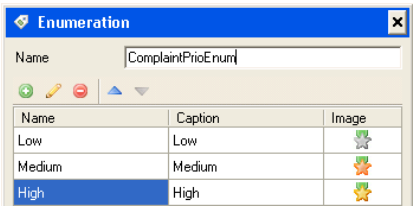

Figure 2, Connecting attribute values to image representations

3.2 Creating the Forms

To define the user interface of the system we create forms in the Mendix business modeler. The editor for forms is similar to other form editing tools, such as those found in Visual Studio .net and Visual Basic. This means that the business modeler supports basic input fields, selectable drop-down list, tables etc. Window layout is supported by the use of layout tables similar in use to the use of table based HTML layouts.

For the example case we create two forms. A *Customer Overview Form* lets the user see a list of Customers (Figure 3). A *Customer Form* lets the user edit customer data, and add complaints that a customer has (Figure 4).

Form-to-object relations

In order to populate the form with information in run-time, the created forms need to be related to the "meta-model". A simple relation is to connect a field to an attribute in one of the objects in the meta-model. A more interesting relation between the forms and the objects is how the mapping between associations and forms is done. As can be seen in the Customer form (Figure 4) a single form can be mapped to

several objects, in this case both the Customer and the Complaint object. In order to indicate that only the Complaints belonging to a certain Customer should be displayed in the Complaint table the Complaint table is put *inside* the Customer sub-form (see Figure 4). This indicates that when interpreting the model and executing a database query the business server should use the association between the Customer and Complaint to find the relevant Complaints and display these in the Complaint table. If the Complaint table was to be put outside the Customer sub-form all Complaints would have been displayed.

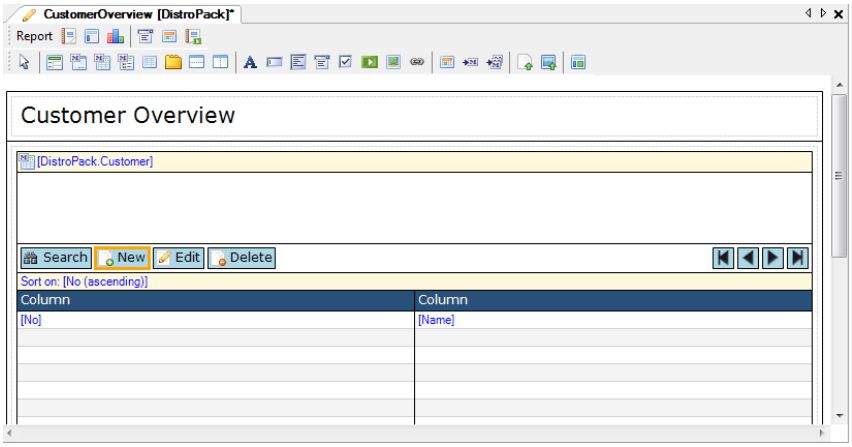

Figure 3, The Customer Overview form in the form editor

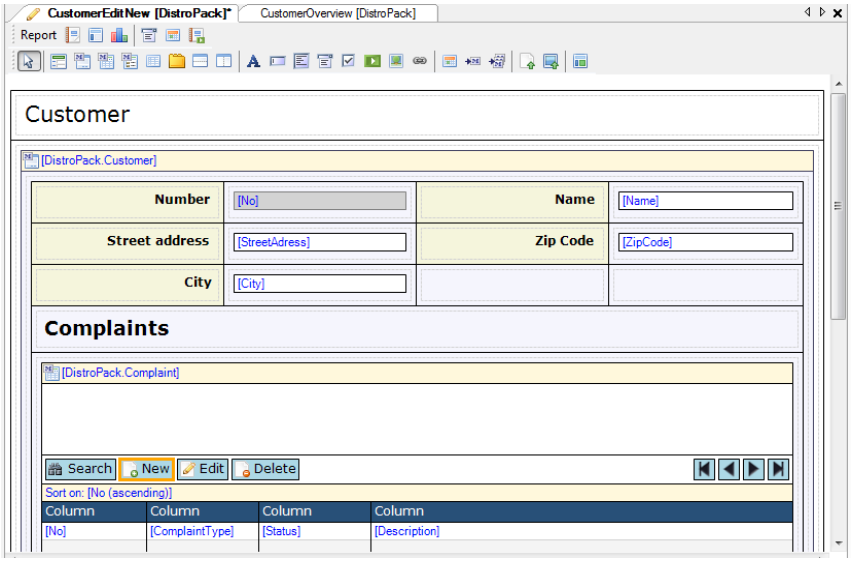

Figure 4, The Customer form in the form editor

Form-to-form relations

A form can also be related to other forms. For example, the Edit and New buttons on the Customer overview form (Figure 3) can be connected to the Customer form. In fact, if we forget to connect the Edit and New buttons to forms an error will be shown in the error list of the modeler (Figure 5). When the user selects a Customer in the overview form and clicks "Edit" the business server will bring up a Customer form displaying the selected Customer. Note that the initialization of the Customer form with the data from the selected customer is done automatically by the business server. This functionality is possible because the server and client keep a stack of selected objects at runtime. The selected objects are also available from the logic defined in microflows.

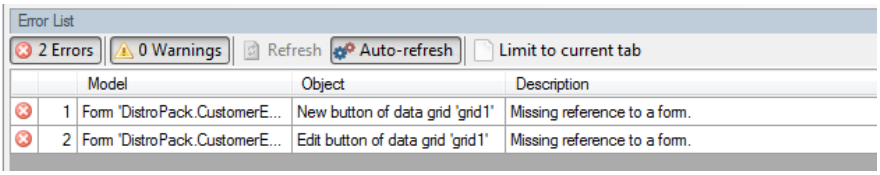

Figure 5, Errors in the models are continuously shown in the error list.

Form-to-microflows relations

Just as events in the meta-model can trigger microflows, events in the forms can also be made to start microflows. For example, it is possible to add a button to a form and connect that to a microflow. The microflow can change objects and these changes can be displayed instantly in the user interface. The simple user-interface logic, such as connecting the overview form to the Customer form can be done without the use of microflow. However, for specific logic, such as displaying different forms depending on the entered data, microflows give full control over the user interface. An example microflow will be shown in the next section.

3.3 Creating Microflows

Microflows are used to define complex logic in system built using Mendix. As mentioned previously, microflows can be triggered by events in both the meta-model and in forms allowing the designer to extend the behavior of the system beyond what can be done using the forms and meta-model. By using a microflow the designer can change objects, and control how and when forms are displayed. Microflows also are a source for integration with external systems using web-services.

For the example system, we use a microflow to set the priority of Complaints. Every Complaint caused by "Damage" shall have prio "High", Complaints due to "Delivery Time" shall have prio "Medium", while all other Complaint causes should have prio "Low". This kind of logic can be expressed in a microflow (Figure 6).

The input to this microflow is a Complaint object, which is specified below the input symbol in the top-left corner of the figure. At the start of the flow is an "Exclusive split" that splits the flow based on the value of the complaint type. The

following activities modify the complaint objects prio attribute. Finally, "exclusive merge" is used to merger the flow into a final end point of the microflow.

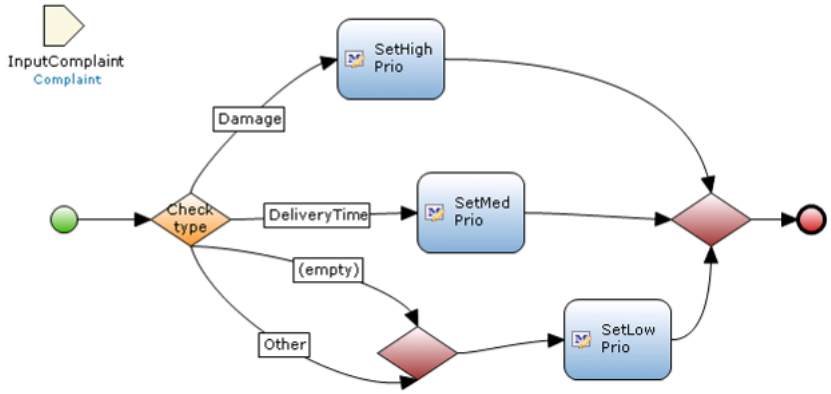

Figure 6, A microflow setting the prioritization of a Complaint

This microflow is very simple, however the editor also allows expressing significantly more complex application logic by the use of loops, calling other microflows, opening forms, etc. Microflows can also call custom made Java code and external web services.

To trigger the start of the microflow we need to connect it to an event, either in the forms or in the objects. For the example case we choose to trigger the microflow whenever a Complaint is saved. To do this, we go back to the Complaint object in the meta-model and define that the "commit" event should trigger the microflow.

Microflow-to-forms relations

As stated before, microflows can be triggered by buttons in the forms. However microflow can also be used to coordinate the usage of forms. For example, it is possible to define a sequence of forms by the use of microflows.

Microflows-to-metamodel relations

Events in the meta-model, such as storing an object or changing it, can be made to trigger microflows. Microflows can also work in the other direction – they can be used to manipulate objects. The full range of CRUD operations (Create-Read-Update-Delete) is supported.

3.4 Running the System

When the models are ready in the business modeller they can be deployed to the business server. At this stage the models are transferred to the model repository, and a database is created based on the "meta-model". When the model has been deployed and the business server is started the system can be accessed via a web-browser. Figure 7 shows the Customer form running in the Internet Explorer browser. The server generates HTML, CSS and JavaScript, thus, there is no need to install any extra plug-ins in the browser.

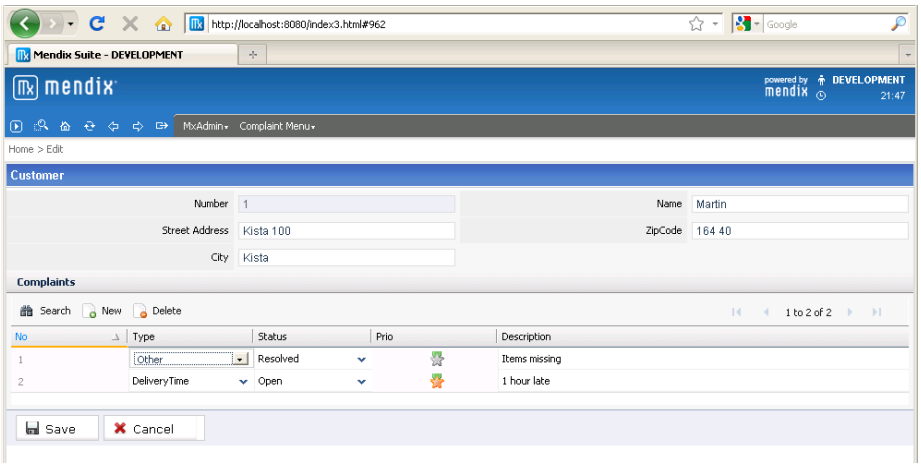

Figure 7, The created Customer form running in a web browser

Note that the "prio" column in the Complaint table (see figure 7) is showing the defined images (see figure 2) for the prioritization level. Due to the triggering of the microflow, the image is updated whenever the Complaint type is updated.

4 Analysis of the Functionality of Mendix

As seen in the previous section a strength of Mendix is that it is quite straight-forward to create a simple application, without the need for coding. In this section we take a closer look at how well Mendix supports the key functionality areas of MDD tools as discussed in section 2.

4.1 Modeling Support of Mendix

Abstraction – the ability to enable work on the correct abstraction level for the given task. A benefit with Mendix is, as shown in the example case models (also counting the forms as models), that the models can be kept on a high level of abstraction compared to what is being achieved by the system. For example, technology issues such as the system architecture, the use of JavaScript, the data storage in database are all abstracted away. On the outset, this might seem like a trivial thing to achieve. However, developers that have worked on creating web systems with support for multiple web browsers can testify that abstracting away technology differences is a non-trivial task.

When it comes to abstraction Mendix also has its drawbacks that the simple case used in this paper did not expose. Most notable is that for complex queries there is a need to specify queries using the XML query language XPath. For example, to retrieve all Complaints that are of type "Damage" in a microflow, there is a need to write a short XPath statement. Moreover, if the need of the user interface goes beyond what is supported by Mendix there is a need to either integrate with an external

system, or to extend Mendix using custom Java code. In addition to reducing the abstraction level of the work, the need to master these technologies increases the competence requirements on the developers.

Understandability – the ability of the models to be understood by stakeholders. A benefit with Mendix is that most models in Mendix are easy to understand – with some introduction. Our experience from using Mendix, and letting students use Mendix, is that the forms and the microflows are easy to understand. However, a drawback is that the notation used for the associations in the meta-model are difficult to fully understand. A big part of the problem here is that for many developers it appears counterintuitive because they have been accustomed to UML class diagrams. This becomes almost a burden as Mendix uses similar graphical notation with different semantics. A similar problem is the use of "meta-model" and "object" in Mendix, as these terms mean something different in other approaches and tools.

Executability – the ability to work incrementally to examine system properties by experimentation or formal analysis. The ease of model execution is a clear benefit of Mendix. The models in themselves are executable by the business server, with no need to add extra code. Of course, as mentioned earlier, there might be the need to add XPath queries into the models for advanced functions. However these XPath queries are still an integral part of the models, and thus there is no need to manage these separately.

Model transformation and refinement – the ability to refine models. This is a drawback with Mendix – there is no way to define or change how the models should be interpreted, or to define custom refinements/model transformations. This can be put in contrast to tool such as OptimalJ, which supports the definition of custom model transformations. To be fair, support for model refinement is probably not in line with the overall design of the tool. Mendix is simply streamlined to provide one level of models that are "just enough" to build systems. Adding model levels would reduce the simplicity of the tool. Mendix do provide support for ensuring syntactic correctness both within a single mode and between the types of models. Considering the specifics of its notation more support for completeness is however desired. E.g. a natural language representation of the facts expressed in the model would assure the modelers that they are using the notation correctly.

4.2 Process Support

Observability – the ability of the tool to clearly show errors in the models. In this area Mendix gives clear benefits. The example shows the business modeler displaying errors when models are incomplete and thus cannot be executed (Figure 5 shows a sample error list). Overall the errors are informative and clearly speed up the development process. Most errors in the list can be double-clicked to display the model that causes the problem, or the settings windows where the error can be fixed.

Collaborative development support. In this area Mendix has both drawbacks and benefits. Mendix modeler does support the simultaneous users working on the models using versioning and locking mechanisms. However, the drawback is that there is no support for advanced branching or merging of branches. It can also be stated that the target for the use of Mendix is not to build very large scale systems. At its homepage,

the company rather points towards the use of Mendix as a complement to other large-scale systems such as ERP systems from SAP and Microsoft.

Turn-around-time – the ability quickly implement and test a feature. Mendix is quick when it comes to deploying and running an implemented feature. Thus the turn-around time is low. A change to the example system described in this paper can be re-deployed and run within 10 seconds. In comparison, a similar simple example system took about two minutes to deploy using OptimalJ. However it must be stated that we have not tried to deploy large systems consisting of hundreds of objects and complex microflows. As pointed out above, this might not be the target area for the use of Mendix to start with.

Integration – the ability to integrate with other systems. Mendix has clear benefits in this area – through the use of microflows it is simple to integrate with external systems. The integration is two-ways, a microflow can call external web services and a microflow can also be published as a web service in itself. Moreover, Mendix provides an open API to the business server, allowing external systems to create, read, update and delete all of the objects in the meta-model. This API can be called using both XML messages and plain Java calls.

Developer competence support – the ability to let others than software developers create systems. In this area Mendix presents a dilemma. On the one hand, for a simple system, such as the example case in this paper, little of software development knowledge is needed. There is a need to understand the basics of information modeling and process modeling. On the other hand, a drawback is that as soon as the system becomes a little more complex additional development skills are needed. E,g,, the use of XPath, and the notion of commits and rollbacks might not be the first focus of a business focused designer. Here it is interesting to note that some of most knowledgeable Mendix consultants have no formal background in information systems. Rather they have developed a great skill when it comes to transfer business needs into something that can be build using Mendix. However there is still a lot more to be done in supporting business experts to review the models. E.g., the microflow has the potential to be an effective communication medium between business experts and model designers, which essence are developers of the system.

Reuse – The ability to aid the reuse of software systems. Support for reuse exists in both the Mendix modeler itself, and also in the intension and services offered by the Mendix company. The tool itself supports reuse in the form of *modules*. Each module contains meta-models, forms and microflows. If properly used to structure a system, each module could be a reuse artifact that can be shared between projects. The Mendix company is also encouraging reuse in the form of a "Mendix App Store" where developers can upload reusable components. While these mechanisms appear to be adequate, more investigation into their effectives is needed.

5 Discussion

The Mendix tool, as all software development tools, has its drawbacks and benefits. We believe that one of the key benefits of Mendix is that its models are placed on a single abstraction level which allows modeling and reasoning on the business level

and still leaves reasonable control over how the final system should behave. Venturing into more abstract levels, or providing multiple abstraction levels, might, for example, leave out the detailed control of the user interface design and behavior. Overall we find the different types of models in Mendix well selected and well integrated. In table 1, we summarize the drawbacks and benefits we found in the key areas listed in Section 2. By reading the drawback column we can get a pointer towards the areas where Mendix is clearly not the choice. It is not a requirement engineering tool per se (lack of abstraction, refinement). It is not suitable for building large scale applications where there is a need for total control of the details (limited collaborative development effort, abstraction).

Table 1, Summary of benefits and drawbacks

	Benefits	*Drawbacks*
Abstraction	Allows user to focus on the business objects and application logic.	Some knowledge of XPath needed to formulate advanced queries.
Understandability	Simple notation and semantics	Substantially different from UML.
Executability	Models are directly executable by the business server.	
Model transformation and refinement	Support for ensuring syntactic correctness.	No support for defining abstraction levels, custom transformations or for helping ensuring completeness.
Observability	Displays errors and guides the user to the model that causes it.	
Collaborative development effort	Support for simultaneous users.	No support for branching and merging of branches.
Turn-around time	Short	
Integration	Allows using external web services, XML messages and Java calls.	
Developer competence support	No programming knowledge needed to implement simple systems.	Some development knowledge needed to develop more advanced systems.
Reuse	Possibility to define and share reusable artifacts via an “App Store”.	

6 Concluding Remarks

In this paper we have analyzed the functionality of the Mendix tool according to set of desired functionality areas derived from literature sources. The analysis suggests that this tool can be highly efficient for developing web-based information systems of simple to medium complexity. Mendix appears to be of good value in small projects with short delivery times, few developers, and development cycles that involve frequent user feedback.

The advancement of MDD tools will continue which lead to more people in organizations developing information systems. From the point of view of business informatics research this poses the future research to concentrate on the role of MDD tools in “non-IT” organizations, the competence development issues of MDD tool users, as well as developing functionality suitable for users with little or no formal IS development education.

In this paper we defined key functionally areas in order to describe the Mendix tool. This list of functionality areas are the subject of further structuring and research.

Aknowledgements

The authors would like to thank Mendix Netherlands and Mendix Sweden for providing us with the ability to use the tool.

References

[Ha06] Hailpern B., and Tarr,P., (2006) Model-driven development: The good, the bad, and the ugly, IBM Systems Journal, vol. 45, no. 3.

[Kl03] Kleppe A., Warmer J., Bast W., (2003) MDA Explained, TheModel Driven Architecture: Practice and Promise, Addison-Wesley, Boston, MA.

[Kr06] Krogstie, J., Sindre G., and Jørgensen H., (2006) Process models representing knowledge for action: a revised quality framework, European Journal of Information Systems (2006) 15, 91–102

[La09] Lano, K., (2009), Model-Driven Software Development With UML and Java, Course Technology, ISBN 978-1844809523.

[Ma05] MacDonald, A., Russell, D., Atchison, B.: (2005)Model-Driven Development within a Legacy System: An Industry Experience Report. Australian Software Engineering Conference p.14-22

[Me02] Mellor, S.J. Balcer, M., (2002). Executable UML: A foundation for model-driven architecture. Addison Wesley. ISBN 0-201-74804-5

[Mo03] Moody, D.L., and Shanks G., (2003) Improving the quality of data models: empirical validation of a quality management framework, Information Systems (IS) 28(6):619-650, Elsevier.

[No09] Norton, D., (2009) "Cool Vendors in Application Development, New Tools, 2009", March 30th , Gartner.

[OM03] OMG-MDA – The Object Management Group (2003), MDA Guide, Version 1.0.1, OMG Document omg/2003-06-01.

[Ra04] Raistrick, C. (2004). Model driven architecture with executable UML. Cambridge: Cambridge University Press.

[Ri08] Ricker J., (2008) Strategic Objectives and Advantages of Model Driven Development, Eclipse Zone, http://eclipse.dzone.com/articles/strategic-objectives-and-advan?page=0,0, accessed 2010-05-06

[Ri96] Rios E., Bozheva T., Bediaga A., Guilloreau N., (2006) MDD Maturity Model: A Roadmap for Introducing Model-Driven Development. ECMDA-FA 2006: 78-89

[Se03] Selic., B., (2003) The Pragmatics of Model-Driven Development, IEEE Software, Vol. 20(5), Sept.

[St06] Stahl, T., Völter M., (2006) Model-Driven Software Development Technology, Engineering, Management, John Wiley and Sons, Ltd., Chichester, England, ISBN: 0470025700.

[Te09] Teppola S., Parviainen P., Takalo J., (2009) Challenges in Deployment of Model Driven Development. ICSEA p.15-20, IEEE

[Uh08] Uhl, A., (2008) Model-Driven Development in the Enterprise. IEEE Software 25(1): 46-49