Integrating Platform Selection Rules in the Model Driven Architecture Approach

Bedir Tekinerdoğan[1], Sevcan Bilir[2] and Cem Abatlevi[2]

[1] TRESE Software Engineering Group, Faculty of Computer Science,
University of Twente, P.O. Box 217, 7500 AE, Enschede, The Netherlands
bedir@cs.utwente.nl
[2] Department of Computer Engineering, Bilkent University,
06800 Bilkent Ankara, Turkey
{sbilirm, abatlevi}@cs.bilkent.edu.tr

Abstract. A key issue in the MDA approach is the transformation of platform independent models to platform specific models. Before transforming to a platform specific model, however, it is necessary to select the appropriate platform. Various platforms exist with different properties and the selection of the appropriate platform for the given application requirements is not trivial. An inappropriate selection of a platform, though, may easily lead to unnecessary loss of resources and lower the efficiency of the application development. Unfortunately, the selection of platforms in MDA is currently implicit and lacks systematic support. We propose to integrate so-called platform selection rules in the MDA approach for systematic selection of platforms. The platform selection rules are based on platform domain models that are derived through domain analysis techniques. We show that the selection of platforms is important throughout the whole MDA process and discuss the integration of the platform selection rules in the MDA approach. The platform selection rules have been implemented in the prototypical tool *MDA Selector* that provides automated support for the selection of a platform. The presented ideas are illustrated for a stock trading system.

1 Introduction

One of the key motivations for Model Driven Architecture (MDA) is the existence of too many platforms, and too many conflicting implementation requirements, reducing the interoperability, portability and reuse of the applications [13]. To this end, MDA explicitly separates the functionality from platform specific concerns and provides Computation Independent Models (CIMs), Platform Independent Models (PIMs), Platform Specific Models (PSMs) and the code (model). One of the key issues is then the transformation among these models. In general, the transformations concern the mapping from PIM to PSM, from PSM to PSM, and from PSM to code. Several transformation techniques have been proposed between the various models and this is actually one of the active research topics.

As such, the development of a system in MDA starts with defining the computation independent model, which is mapped to a platform independent model, and by a series of transformations gradually the platform specific properties are included through the platform specific models, eventually resulting in the final code.

U. Aßmann, M. Aksit, and A. Rensink (Eds.): MDAFA 2003/2004, LNCS 3599, pp. 159 – 173, 2005.

Although, the mapping to different models and the related transformations have gained more interest, the selection of particular platform is not explicitly addressed. During the last years, different platforms have been proposed such as CORBA, .NET and J2EE. Each project may have its own requirements and constraints and depending on the project parameters, different types of platforms may be required. It is important that the right platform is selected to meet the project requirements and to avoid unnecessary loss of resources because of maintenance problems later on. Selecting an inappropriate platform will require redoing the whole transformation process between the different models including PIM to PSM, PSM to PSM and PSM to code.

Selecting a platform, however, is not a trivial process. Each platform usually addresses different properties and selecting a platform requires a broad understanding of the available platforms. Currently, in MDA the selection of platforms is basically implicit, and no systematic support is provided to guide the software engineer in selecting the right platforms.

We propose to integrate so-called *platform selection rules* for selecting an appropriate platform in the MDA approach. Platform selection rules are derived from the *platform domain model*. The *platform domain model* defines the commonality and variability of a set of platforms and is derived using domain analysis techniques. The platform selection rules help to determine to which extent the platform is suitable or not.

The approach is generic, yet as an example we define the rules for selecting .NET and J2EE platforms. We illustrate our ideas for a stock trading system and describe a prototypical tool *MDA Selector,* which implements the platform selection rules.

The remainder of this paper is organized as follows: Section 2 introduces the example case stock trading system that is used throughout the paper to discuss the problems and the solutions. Section 3 provides the background on transformation rules and additionally introduces the notion of platform selection rules. Section 4 discusses the approach for extracting and specifying the platform selection rules. Section 5 discusses how platform selection rules can be integrated in the MDA approach. Section 6 presents the prototypical tool that implements the platform selection rules for J2EE and .NET. Section 7 provides the related work and finally section 8 presents the conclusions.

2 Example: Stock Trading System

Development of a system in MDA proceeds from CIM to PIM, from PIM to PSM, and from PSM to code. In the following, we will show the CIM and the PIM for a stock trading system and then discuss the motivation for systematic selection of platforms.

2.1 Computation Independent Model

In the stock trading system, the client requests the stockbroker to enter a buy or sell order for a certain number of stocks. An order results in a deal when a matching bid of the opposite type is present. The system automatically performs the possible deals and entails several bookkeeping actions.

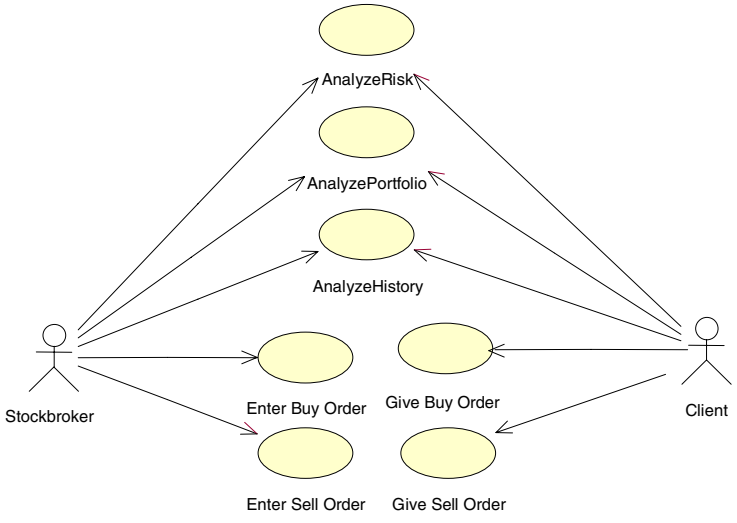

Fig. 1. Computation Independent Model for Trading System (Business Model)

Figure 1 represents (part of the) computation independent (business) model of the stock trading system. In the use case, there are three two actors: *StockBroker* and *Client*. The actor *StockBroker* performs the use cases *Analyze Risk, Analyze History, Analyze Portfolio, Enter Buy Order, Enter Sell Order*. The actor *Client* can apply the use cases *Analyze Risk, Analyze Portfolio*, *Analyze History*, *Give Sell Order* and *Give Buy Order*.

2.2 Platform Independent Model

The CIM does not include any computational issues and defines the solution from a requirements and business perspective. The PIM provides a model of the application including the computational aspects but refraining from the platform specific aspects. Fig. 2 shows the (simplified) PIM for the stock trading system.

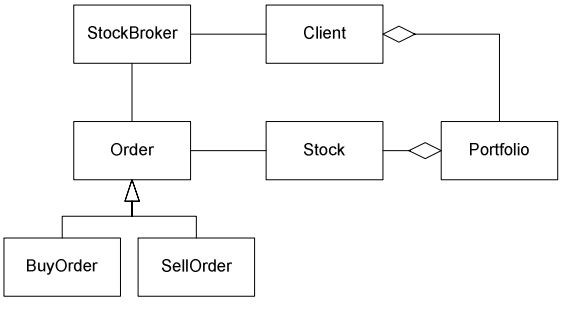

Fig. 2. (Simplified) PIM for stock trading system

2.3 Selection of Platforms

The representation of the platform independent model is important to support the quality factors of reuse, interoperability and portability to different platforms. However, for the more concrete implementation it is necessary that a platform is selected after which the PIM is mapped to a PSM including the specific properties of the selected platform.

For the stock trading system, the first important question is then which platform to select. There are various platforms and it is not trivial to select a platform that best fits the needs of the stock trading system. All of the existing platforms have different properties and in principle can be selected to realize the PSM. Albeit any changes to the platform will not influence the PIM in the MDA perspective, the selection of a given platform will have a serious impact on the platform specific model. If a non-optimal platform is selected this will directly impact the PSMs which need to be generated again. If the right transformation rules exist, and if these are automated then the generation of PSMs might be better supported. Nevertheless, it is not efficient to continuously rely on a trial-and-error approach until the right platform has been selected, and likewise it is worthwhile to provide a systematic approach, which supports the decision on a platform. Unfortunately, this is not explicit in MDA yet. The following sections elaborate on this issue.

3 Transformation Rules and Platform Selection Rules

Several approaches have been proposed for mapping PIM to PSM, such as use of templates, marks, and patterns. We can categorize all these approaches as *transformations*. Within this context, Kleppe et. al. provide the following definitions [10]:

Transformation is the automatic generation of a target model from a source model, according to a transformation definition.

A **Transformation Definition** is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language.

A **Transformation Rule** is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language.

All these definitions and tools are primarily focused on *transformation* of the models down to code. Although MDA improves the interoperability and portability of the systems, it does not explicitly define which platform to choose for a given set of project requirements, though. In fact, this is actually the strength of MDA; it does not commit to a particular platform.

Nevertheless, sooner or later a platform must be selected to realize the system. Since the selection of the platform is not explicit this is usually done in an informal and less systematic manner.

Complementary and in alignment to the above definitions we introduce the definitions that are required for selecting platforms:

Platform Selection is the automatic selection of a platform according to the input from the application requirements.

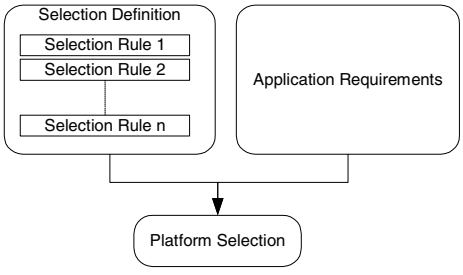

Fig. 3. Platform selection inputs

Platform Selection Definition is a set of selection rules that together describe the selection of platforms,

Platform Selection Rule is a description on the selection of a particular platform based on a given property.

The idea of selecting platforms is given in Fig. 3.

The rules for selecting platforms are different from existing transformation rules in two perspectives. First, the rules are defined before the transformation rules. Second, the rules do not transform any model but only support the system designer in the selection of the platform. Altogether, we think that these platform selection rules are complementary to the existing transformation rules.

4 Approach for Defining Platform Selection Rules

Intuitively, it seems sound to support the software engineer in selecting a platform based on a given set of rules. The question of course is how to define these rules. For this, we propose to apply domain analysis techniques. In section 4.1, we will discuss the approach for defining a *platform domain model* using domain analysis techniques. Based on this platform domain model, the approach for deriving *platform selection rules* will be explained in section 4.2. Finally, section 4.3 discusses the selection of platforms based on the defined rules and the project constraints.

4.1 Defining a Platform Domain Model

Domain analysis can be defined as the process of identifying, capturing and organizing domain knowledge about the problem domain with the purpose of making it reusable when creating new systems [1]. Domain analysis focuses on a given domain and aims to represent this domain in a reusable format. The UML glossary provides the following definition of the term domain [8]:

Domain: An area of knowledge or activity characterized by a set of concepts and terminology understood by practitioners in that area.

Conventional domain analysis methods consist generally of the activities *Domain Scoping* and *Domain Modeling* [1]. *Domain Scoping* identifies the domains of interest, the stakeholders, and their goals, and defines the scope of the domain. *Domain Modeling* is the activity for representing the domain, or the *domain model*.

The domain model can be represented in different forms such as object-oriented language, algebraic specifications, rules, conceptual models etc. Typically a domain model is formed through a commonality and variability analysis to concepts in the domain.

Our focus in this paper is on modeling platforms for reuse. The MDA Guide provides the following definition for platform [13]:

Platform*:* a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented.

The MDA guide further classifies platforms into *generic platform types, technology specific platform types* and *vendor specific platform types*. The discussion of our study is independent of these classifications.

A domain for our purposes represents the area of knowledge on the set of platforms that we are interested in. We term this as the **platform domain model.** Related to this, in the MDA Guide the notion of platform model is defined [13]:

Platform model provides a set of technical concepts, representing the different kinds of parts that make up a platform and the services provided by that platform.

This definition focuses implicitly on the modeling of a single platform. With *platform domain model,* we define a model that represents one or more platforms. For this, it is required to model the common properties and the variant properties of the corresponding alternative platforms. To this end, we apply *feature modeling,* which is a well-known technique in domain analysis [6]. Feature modeling results in a *feature model*, which consists of a feature diagram and additional semantic information such as descriptions of features, rationale of features, etc. A feature diagram represents a hierarchical representation of the features of a system. The root of a feature diagram represents a concept.

Fig. 4 presents the approach for modeling platforms. In the first step, it is decided which platforms one is interested in and the corresponding domains are identified. This is actually the domain scoping for platforms. As an example, one might decide to focus on *Corba, .NET* and *J2EE*. Once the platforms are known, the corresponding platform domain model will be developed. An appropriate platform domain model that meets the application requirements might already exist in the literature. If no suitable platform model exists then this is defined using commonality and variability analysis to the knowledge sources on the corresponding platforms. The knowledge sources might include textbooks, technical papers, human experts or systems, which implement the corresponding pattern. Once the platform domain model is developed it will be evaluated based on the application requirements and the platform information. If the evaluation is passed then the platform domain model can be utilized.

Fig. 5 presents, for example, a feature diagram for platforms as a result of domain analysis to J2EE and .NET platforms. It describes a platform as consisting of *Vendor, Operating System, Architecture, Language* and *Services* features. This feature model has been derived after a commonality and variability analysis to knowledge sources on .NET and J2EE [14][15] [16].

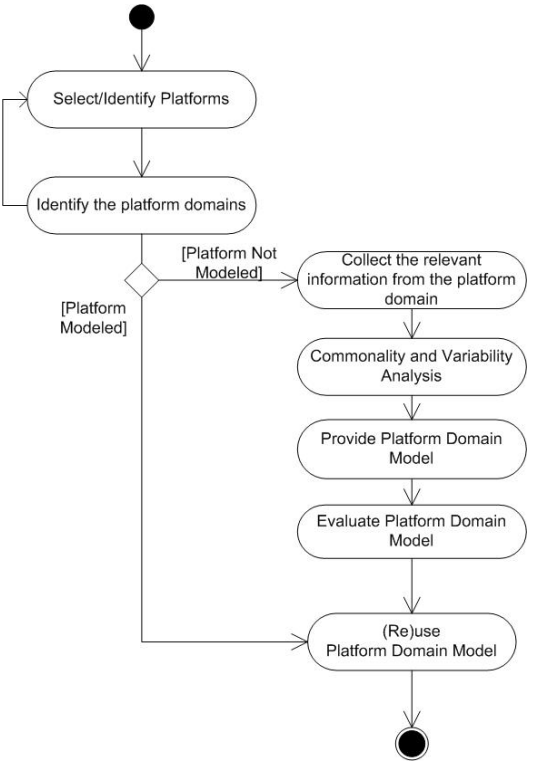

Fig. 4. Process for deriving platform domain model

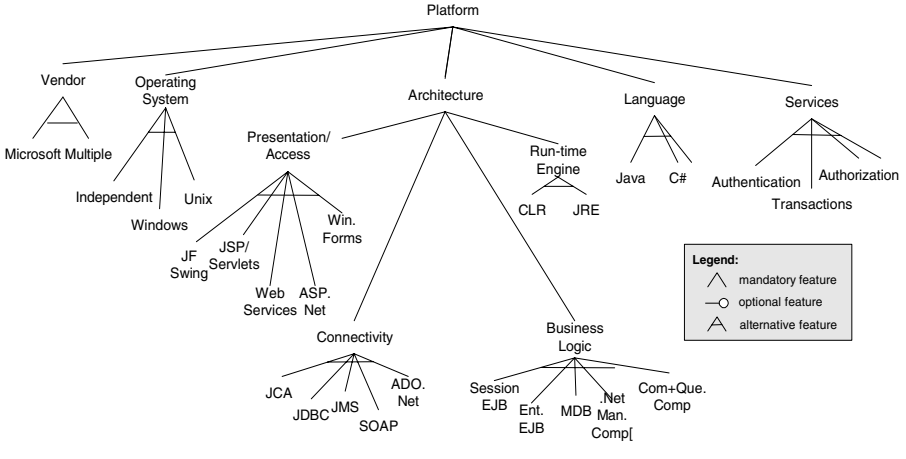

Fig. 5. Feature Diagram for Platform Domain

Table 1. Properties for .NET Platform

P1. Vendor is Microsoft
P2. Operating System is Windows
P3. Presentation Access is ASP.Net, Windows Forms, Web Services
P4. For Database Connectivity ADO.Net and SOAP is used.
P5. Business logic is provided through .NET Managed components and COM Queued components
P6. Requires Common Language Runtime (CLR) run-time engine.
P7. Source code is written in C#.
P8. Supports transaction and authentication services
P9. ….

Table 2. Properties of J2EE Platform

P1. Vendor is independent (more than 30)
P2. Operating System is independent
P3. Presentation Access is JSP, JFC, Web Services
P4. For Database connectivity Java Database Connectivity (JDBC) protocol, Java Connector Architecture (JCA), Java Messaging Service (JMS) and SOAP is used.
P5. Business logic is provided through Session Enterprise JavaBeans, Entity Enterprise JavaBeans and Message Driven Beans.
P6. Requires Java Runtime Engine (JRE)
P7. Source Code is written in Java
P8. ….

For deriving *platform selection rules*, we represent the platform domain model as a set of *platform properties*. A platform property is defined as a description of the feature of a platform and as such, is directly derived from the feature diagram. For example, Table 1 and Table 2 represent (a set of) properties for .NET and J2EE platforms, which have been derived from the feature diagram in Fig. 5.

4.2 Extracting the Rules from Platform Domain Model

Once the platform domain model has been derived it can already be manually utilized in selecting the appropriate platform. For automating the rules a further formalization is required. We do this by mapping the properties to the platform selection rules. The platform selection rules are expressed using conditional statements in the form IF <condition> THEN <consequent>. For example property P1 in Table 1 and Table 2 lead to the rules R1 and R2, respectively in Table 3. Note that the list is not comprehensive due to space limitations.

4.3 Selecting Platforms Using Application Constraints

The platform selection rules represent the general cases for selecting platforms. For selecting a platform we need to define the corresponding application constraints as it was discussed in section 3 and illustrated in Fig. 3. Each constraint can trigger a rule in the rule definition. As such, for a given set of constraints, a set of rules will be triggered. The triggering of a rule means that the condition requested by the constraints matches the condition of the platform selection rule. Assume that, for example, the constraints as defined in Table 4 are specified for the stock trading system.

Table 3. Heuristic Rules for Platform Selection for J2EE and .NET

R1.	IF the vendor should be independent THEN select the platform J2EE
R2.	IF the vendor should be Microsoft THEN select the platform .NET
R3.	IF the platform should be independent from the operating system THEN select the platform J2EE
R4.	IF the platform should have Windows operating system THEN select platform .NET
R5.	IF JVM run-time engine is installed/required THEN select the platform J2EE
R6.	IF CLR run-time engine is installed/required THEN select the platform .NET
R7.	IF the application will be implemented in Java THEN select the platform J2EE
R8.	IF the application will be implemented in C# THEN select the platform .NET
R9.	IF transaction and authentication support is required THEN select the platform J2EE
R10.	IF database access with JDBC is required THEN select the platform J2EE
R11.	IF database access with ADO.NET is required THEN select the platform J2EE
R12.	IF ASP.NET is required as a web-tier component THEN select the platform .NET
R13.	….

These constraints trigger five rules R3, R6, R7 and R9 in Table 3. This leads to an indecisive result to select J2EE (for R3, R7 and R9) and .NET (for R6). As in this case, very often the application requirements do not lead to a single possible platform. The reason for this is, firstly that the corresponding platforms share some common properties, and secondly, the application requirements might be conflicting itself. To support the decision process in case of conflicts, we apply the prioritization of the constraints by assigning each of these a value between 1 and 9. Hereby the value 1 is defined as a supportive but least important constraint, whereas 9 represent a very strong decisive constraint. Note that the constraints C1 to C4 in Table 4 correspond to the elements in the feature diagram as defined in Fig. 5. In principle, it would be possible to annotate the priorities to the feature diagram as well. On the other hand, the priorities for each project might change and in that sense, it is more appropriate to separate the priorities from the feature diagram.

The priority values are assigned to the triggered rules. The decision for each platform depends then on the number of fired rules and the values of the constraints. Therefore, for the constraints in Table 4 this means that the total score for J2EE is 9+8+8=27 and the score for .NET is 5. This information could be used for the final decision or for a closer look at the conflicting requirements. In fact, the prioritization and the policy for selecting platforms based on these scores might be refined. What is important here is that this decision is made explicit.

Table 4. Constraints and Priorities for Stock Trading Application

Constraint	*Priority*
C1. The application should work in all environments so the platform must be operating system independent.	***9***
C2. The language which will be used for implementation must be in Java	***5***
C3. The run time engine should be CLR.	***8***
C4. Transactions and authentication are required.	***8***

5 Integrating Selection of Platforms in the MDA Pattern

Fig. 6a illustrates the integration of the platform selection rules in the MDA pattern. The drawing builds on the pattern as defined in the MDA Guide [13]. The rectangles represent either the platform independent models or the platform specific models, the arrows represent transformations and selections. In fact, the selection of platforms appears to be complementary to the MDA pattern. In the current MDA pattern the selection is implicit. Fig. 6a makes this explicit by adding an operation which selects (and models) the platform. Similar to the initial MDA pattern the drawing is intended to be suggestive and generic. The platform independent model together with the selection of platform and the corresponding information on the platforms are combined to produce a platform specific model. There can be many ways in which transformations may be done. The selection is based on the approach as defined in the previous sections.

It should be noted that the terms PIM and PSM are just relative terms and it is difficult to draw a strict line between platform independent and platform specific model. In fact, a platform specific model can function as a platform independent model for a next stage. For example, the upper PIM that is independent of many platform choices, could be mapped to a PSM which is specific to middleware platforms. However, the transformation could be carried out so that the PSM is independent of the particular component platforms.

This can also be derived from the given example case. The original platform independent model is first mapped to a J2EE platform specific model, which remains independent of the choice of a particular component platform in J2EE. In the given example case, the J2EE-specific model can thus be considered as a PIM as well. There are three basic component platforms in J2EE: JSP (Java Servlet Pages), Servlet and EJB (Enterprise Java Beans). Before transforming the J2EE specific platform independent model, we have to select the specific component platform in J2EE. This process is illustrated in Fig. 6b. Note that the extended MDA pattern as defined in Fig. 6a is applied twice in Fig. 6b.

Selecting the component platforms of the J2EE platform requires defining the corresponding platform selection rules. In principle, this is the same process as defined in the previous sections, and we do not elaborate further on this.

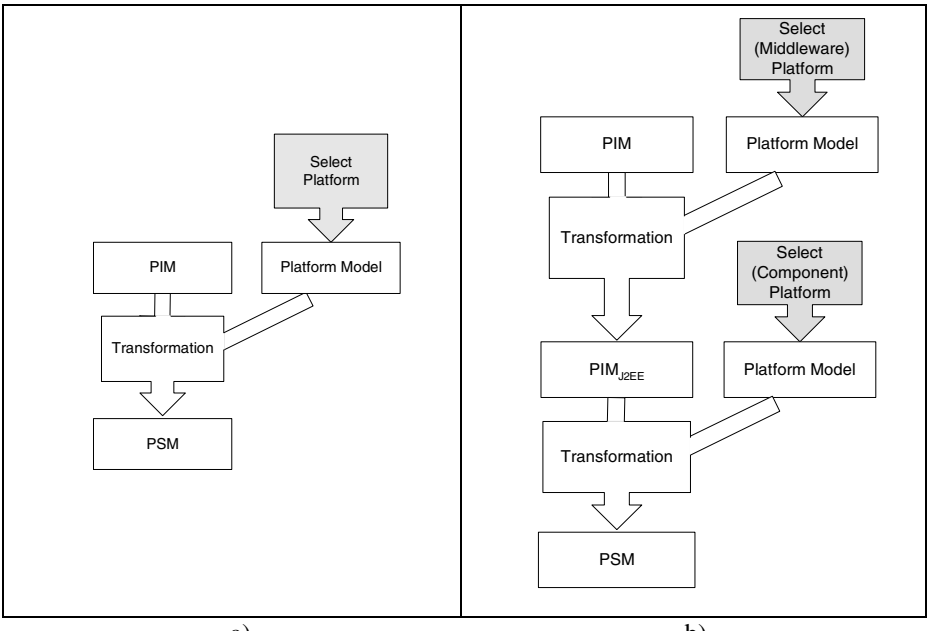

Fig. 6. The integration of platform selection rules in the MDA pattern

6 Platform Selection Tool

Since the platform selection rules have been formalized, they can be easily implemented in a tool to provide automated support for the decision of a platform. We have implemented a prototypical tool environment for selecting a platform for a given PIM. The tool environment is called *MDA Selector*. A snapshot of this tool is given in Fig. 7. *MDA Selector* simply implements the rules that have been derived from the platform domain model. The tool starts by prompting the user in order to determine the middleware platform by using the check boxes, which represents the properties for different platforms. In addition, each property can be assigned a number between 1 and 9. If all the required properties are checked and the numbers to these properties have been assigned, then the user of the tool can click the action button *Decide*, to get the decision on the platform. The decision is shown in the right corner using colored rectangles. The size of the rectangle indicates the degree of preference for the given platform. The rules themselves have been implemented as objects with the attributes *condition, platform* and *value*. The attribute *condition* represents the condition of the rule, the attribute *platform* refers to the selected platform for the condition, and the attribute *value* represents the number assigned to the rule. Upon pressing the *Decide* button the algorithm for selecting the platform is executed. Hereby, the selected properties are matched against the implemented rules. In case a selected property matches the condition of a rule, the rule will be triggered, that is the value for the rule

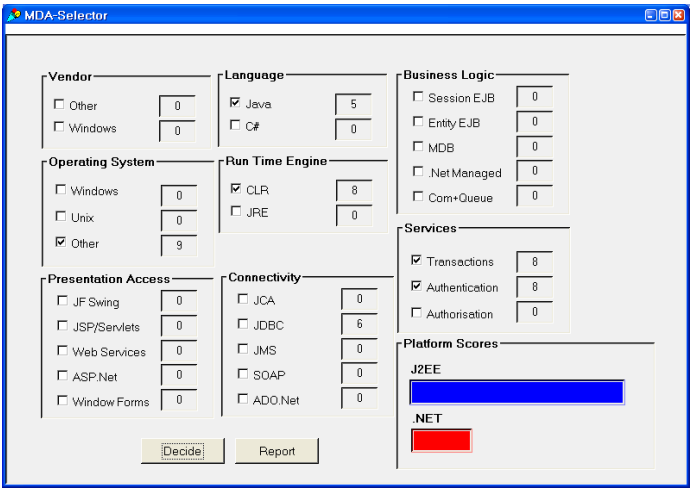

Fig. 7. Platform Selection Tool

is set to the entered value for the property and the degree for the corresponding platform is updated. The action button *Report* provides additional information on the result of the selection. The tool is implemented in Visualworks 3.0 and currently includes a simple, though, effective implementation.

7 Related Work

The MDA guide [13] provides a definition of platform model but no explicit process for deriving the platform model is given. We adopt domain analysis techniques for systematically defining platform models. In [3] and [19], the notion of Platform Description Model is presented, which is similar to our notion of platform model since both are representations of the corresponding platform. In [4] the Platform Model is expressed at a conceptual level and does not specifically represent a formal model. In all of these approaches, the term platform model is utilized in transforming a PIM to a PSM. In our approach, the platform model is used to derive the rules for selecting the platforms. Later on, the platform model can still be used as an input to the transformation process.

In [12] exploration and selection of alternative transformation models using algebraic techniques is presented. Hereby the possible set of transformation models is represented as *transformation spaces.* In our approach, we focus on modeling the heuristic rules for selecting platform models. As such, both approaches seem to be complementary to each other.

In [7] the authors discuss the relation between MDA and a configurable software product line family. Similar to our understanding, the authors state that platform models are at best derived using domain engineering techniques. A PIM in MDA represents the model for a family of platform specific models and as such, seems to perfectly align with the idea of developing domain models in domain engineering. We

have shown how we can derive platform properties from feature diagrams, and platform selection rules from these properties. It would be interesting to investigate the relation between MDA and domain engineering further.

In our previous work, we have modeled heuristic rules for automating software development methods [17] [18]. In these approaches, the rules represented selection, elimination and transformation actions. In this paper, we have utilized rules merely to select a platform. A useful further step would be to integrate both selection and transformation rules in a common tool environment.

The tool that we developed can be considered as an initial expert system that codifies the rules for selecting platforms. An expert system usually consists of a knowledge base (facts), rule base including production rules, and an inference engine for triggering these rules [11]. Expert systems have also been applied for the hardware configuration problem. Hereby, the expert system determines the best hardware configuration based on the rules in the expert system knowledge base as well as the customer requirements.

8 Conclusions

It appears that current research on MDA primarily focuses on transformation of models. Before transforming to a particular platform specific model, however, it is necessary that the appropriate target platform is selected. Currently, the selection of platforms is generally considered an ad hoc issue and largely remains implicit. However, given the currently relatively broad set of platforms, which is despite MDA still expected to grow in the future, it is certainly not a trivial task to select the platform that optimally meets the application requirements. As such, we argue that besides of transformation process in MDA also the selection of platforms should be integrated in the MDA development pattern.

In section 3 we have given the definitions of *platform selection*, *platform selection definition*, and *platform selection rule* as a complementary set of definitions on *transformation*, *transformation definition* and *transformation rule*.

To extract the platform selection rules we have proposed to adopt domain analysis techniques. In this context, we have primarily focused on defining properties of platforms and derived the rules based on these properties. Further, a first prototypical tool environment which indicates the use of the selection of platforms is provided. We have illustrated the approach for selecting a platform for stock trading system.

Since PIM and PSM are just relative terms and a PSM can also function as a PIM the platform specific transformations can be applied at different levels in MDA. Similarly, in section 6 we have shown that this counts for selecting platforms as well. Hereby, first the middleware platform was selected and then the particular component platform in the given middleware.

Although the standard use of MDA assumes that products are built for all platforms, and the transformation is considered as automatic, we have highlighted the selection of platforms to determine whether it is suitable or not. From our study, we can also conclude that the selection of platforms is complementary to the transformation process. We have primarily focused on the platforms J2EE and .NET. Although the presented example application is rather small, we think that the

presented ideas can scale easily for larger applications. This is because the basic complexity for selecting the platforms is mostly defined by the platform domain model itself, rather than the size of the application. In fact, the presented rules are directly derived from the platform model and are more or less fixed for a given platform. The only difficulty for larger applications is that the decision for each rule could be more difficult, but still manageable. In our future work, we will provide the domain models for other platforms as well and derive the rules to support the software engineer in selecting the appropriate platforms.

Acknowledgements

We would like to thank the anonymous reviewers, Klaas van den Berg, and Ivan Kurtev for their valuable feedback on earlier versions of this paper. This research has been carried out in the *Aspect-Oriented Software Architecture Design project*, which is funded by the Dutch Scientific Organisation in the *Jacquard Software Engineering Program.*

References

[1] G. Arrango. Domain Analysis Methods. In Software Reusability, Schäfer, R. Prieto-Díaz, and M. Matsumoto (Eds.), Ellis Horwood, New York, New York, pp. 17-49, 1994.

[2] U. Assmann, Automatic roundtrip engineering, Electronic Notes in Theoretical Computer Science Vol. 82, Issue 5.

[3] J. Bézivin and N. Ploquin, Combining the Power of Meta-Programming and Meta-Modeling in the OMG/MDA Framework, OMG's 2nd Workshop on UML for Enterprise Applications, San Francisco, USA, December, 2001

[4] S. Bilir & C. Abatlevi. Model-Driven Architecture Based on Design Space Modeling. Technical Report, Department of Computer Engineering, Bilkent University, Ankara, Turkey, June, 2003.

[5] S.E. Borch, J.W. Jespersen, J. Linvald, Kasper Østerbye. A Model Driven Architecture for REA based systems. In Proc. of Model Driven Architecture: Foundations and Applications, pp. 103-108, University of Twente, Enschede, The Netherlands, 2003.

[6] K. Czarnecki & U. Eisenecker. Generative Programming: Methods, Tools, and Applications, Addison-Wesley, 2000.

[7] S. Deelstra, M. Sinnema, J. van Gurp & J. Bosch. Model Driven Architecture as Approach to Manage Variability in Software Product Families. In Proc. of Model Driven Architecture: Foundations and Applications, pp. 109-114, University of Twente, Enschede, The Netherlands, 2003.

[8] G. Booch, J. Rumbaugh & I. Jacobson. The Unified Modeling Language User Guide, Addison-Wesley, 1999.

[9] K. Kang, S. Cohen, J. Hess, W. Nowak, & S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report, CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, November 1990.

[10] A. Kleppe, J. Warmer, W. Bast. MDA Explained, The Model-Driven Architecture: Practice and Promise, Addision-Wesley, 2003.

[11] M.R Klein & L.B. Methlie, Knowledge-based Decision Support Systems, 2nd Ed., Wiley, 1995.
[12] I. Kurtev & K. van den Berg. A Synthesis-Based Approach to Transformations in an MDA Software Development Process. In Proc. of Model Driven Architecture: Foundations and Applications, pp. 121-126, University of Twente, Enschede, The Netherlands, 2003.
[13] MDA Guide Version 1.0. Edited by Joaquin Miller and Jishnu Mukerji. http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf, June, 2003.
[14] J2EE and Microsoft .NET, Oracle White Paper, April 2002.
[15] P. Perrone, S.R. & T. Schwenk J2EE Developer's Handbook, SAMS, 2003.
[16] C. Szypersky. Component Software: Beyond Object-Oriented Programming, Addison-Wesley, 2002.
[17] B. Tekinerdogan & M. Aksit. Providing automatic support for heuristic rules of methods. In: Demeyer, S., & Bosch, J. (eds.), Object-Oriented Technology, ECOOP '98 Workshop Reader, LNCS 1543, Springer-Verlag, pp. 496-499, 1999.
[18] B. Tekinerdogan. Formalizing heuristic rules of Extreme Programming. Dept. of Computer Science, University of Twente, 2003.
[19] E.D. Willink. UMLX: A graphical transformation language for MDA. In Proc. of Model Driven Architecture: Foundations and Applications, pp. 13-24, University of Twente, Enschede, The Netherlands, 2003.