Continuous Migration of Mass Customized Applications

Michiel Overeem
dept. Product Development
AFAS Software
Leusden, The Netherlands
michiel.overeem@afas.nl

Slinger Jansen
dept. Information and Computing Sciences
Utrecht University
Utrecht, The Netherlands
slinger.jansen@uu.nl

***Abstract*—Model-driven engineering is a known approach for increasing both the quality and productivity of software development. It also enables business analysts to take a more active role in the development process. The model allows the application to be tailored to the customer, who no longer needs to adjust to the software. Through the model it is possible to mass customize the developed software applications. Mass customization allows software producing organizations to efficiently produce and maintain multiple similar software products. Through model-driven engineering the similarities between these products can be exploited while the variations are managed through models.**

The increase in quality and productivity, and the mass customization is achieved by raising the level of abstraction on which software is developed. Changes done on the higher level of abstraction should be translated into changes on the resulting application. However, in many model-driven systems the migration of the resulting application towards the new intended state is still a (partially) manual and labor-intensive step. This manual step requires engineers to work on a lower level of abstraction, and thus forgoes on the increase of quality and productivity.

In this research we study model-driven engineering environments in which the application is generated from a model. Whenever models or parts of these model-driven engineering environments evolve, the goal is to automatically migrate the application and data as well. In order to reach this goal, we formulate three solution approaches. First of all a categorization of migration triggers that can occur is discussed, to increase the understanding of the context of the migration. Migration triggers can be handled with different migration strategies. Secondly, we discuss the integration of the microservice architecture style to achieve fine-grained incremental migration. Finally, we discuss event sourcing as a software architecture pattern to mitigate the complexity of data migrations. Through these solution approaches we present our work in progress on continuous migration for mass customized applications.

I. Introduction

Model-driven engineering (MDE) is a known approach for increasing both the quality and productivity of software development teams. According to Díaz et al. [1] these improvements are achieved by raising the level of abstraction. MDE tools can also enable business analysts to take a more active role in the development process. The application of MDE, however, is not only positive. Multiple experience reports, such as those by Tolvanen and Kelly [2] and Paige and Varró [3], discuss the challenges and the effort it takes to create MDE tools. Clark and Muller [4] report on two startups around MDE tools and the lessons learned from those startups.

Our research focuses on MDE for enterprise software applications (ESAs). Fowler [5] states "Enterprise applications are about the display, manipulation, and storage of large amounts of often complex data and the support or automation of business processes with that data." According to Gartner [6] examples of ESAs are CRM and ERP software. The characteristics of ESAs are distinct from for instance games and result in different requirements for application migration.

We believe that the model-centric approach of MDE is especially promising for ESAs. Brown [7] defines the model-centric approach as "the system models have sufficient detail to enable the generation of a full system implementation from the models themselves". The first promise is the increase of productivity and quality. ESAs often contain repetitive patterns of functionality, such as the maintenance of data. MDE increases the productivity by generating the software for every instance of these patterns. Not only can MDE increase the productivity by deriving these components from a model, it also increases the quality of those components by doing it consistently.

The second promise is flexibility, or mass customization (as discussed by Krueger [8]) for multi-tenant ESAs. Through the model it is possible to mass customize the developed software applications. Mass customization allows software producing organizations (SPOs) to efficiently produce and maintain multiple similar software products. Through MDE the similarities between these products can be exploited while the variations are managed through models.

The third promise is that of self-service. Not only SPOs want to customize the software, customers (tenants) want to customize their own application to support their business processes. Tenants want to be in control and make the customization themselves.

Multi-tenant ESAs can be developed with MDE through Model-Driven Engineering Environments (MDEEs). As stated in our earlier research (Overeem et al. [9]), MDEEs implement the model-centric approach. One of the challenges in MDEEs is the *continuous migration* of mass customized applications.

This work is a result of the AMUSE project. See amuse-project.org for more information.

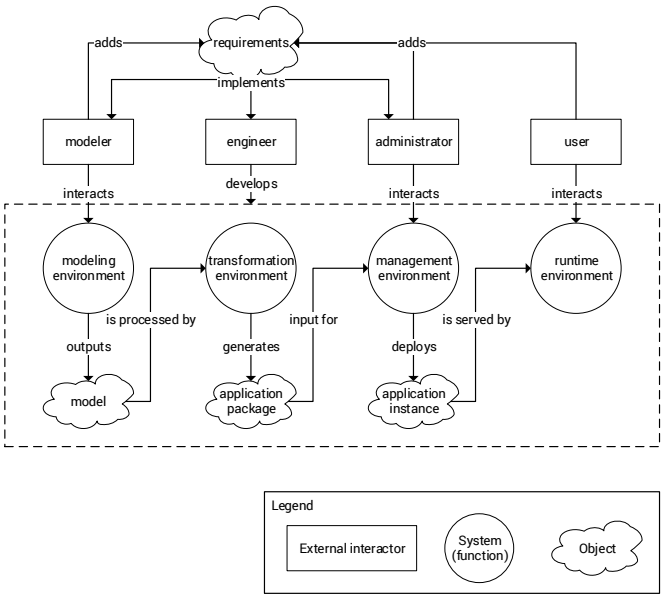

Figure 1. A model-driven engineering environment enables a modeler to create a model in a *modeling environment*. The model is subsequently translated by the *model execution engine* (using a *model execution approach*) into an application. An administrator uses the deployment environment to deploy the application to a runtime environment. The user interacts with the runtime environment.

Migration is defined as the steps that need to be executed to make the actual deployment of an application reflect the desired deployment. Examples of application migrations can be the re-generation of the application to reflect an evolved model, it could be the migration to a new version of the virtual machine, or even the migration to a new cloud provider.

II. Model-Driven Engineering Environments

Model-driven engineering environments (MDEEs) allow the design and develop of software using the MDE approach. While there are different variation points, all MDEEs share three characteristics. First, they serve three key personas: the modeler, the administrator, and the user. The engineer is involved as the fourth persona, but this person is not served by the MDEE, but builds and maintains it. Second, three key artifacts are produced in the MDEE: the model, the application package, and the application instance. Third, an MDEE offers four essential system functions: the modeling environment, the transformation environment, the management environment, and the runtime environment. The MDEE concept is visualized by Fig. 1.

The *modeler* uses the *modeling environment* to produce the *model*. The modeling environment varies between different MDEEs, examples are graphical or text based modeling environments. Versioning, sharing, and collaboration are features that could be offered, but are not essential for the MDEE. The essential function is the production of the model.

The modeler persona represents different types of modelers, we identified four types of modelers in earlier research (Overeem et al. [9]): laymen, technical business users, sql experts, and developers.

The model is processed by the *transformation engine* which generates an *application package*. The application package takes on different formats, for example binaries when code generation is used.

The application package is input for the *management environment*. One of the responsibilities of the management environment is the deployment the application package. Through the deployment the *application instance* is created. The management environment is responsible for managing the application instances. Migration of applications, caused by different triggers, is executed by this environment. The intent of the management environment is to automate this process, however, the *administrator* can also use the management environment to execute manual tasks.

The application instance is executed by the *runtime environment*. This runtime environment consists of infrastructure (such as operating systems and database platforms), services, frameworks, and other components required to execute the application instance. The third persona, the *user*, interacts with this environment to execute the features offered by the application.

III. Requirements for Continuous Migration

We specifically focus on those MDEEs that allow the user to also take the role of the modeler. Without intervention of the SPO and its engineers and administrators, the user should be able to evolve the application through model evolution. This results in more agility for the customer organization, they will not be dependent on the SPO for the continuous development of their application (at least within the capabilities of the model). Therefore our research focuses on four operational requirements for the migration of applications: automation, performance, availability, and safety.

First, **the migration should be automated.** With automated, we mean that the required migration steps should be derived from the migration trigger. At no point should an engineer manually analyze the required migration and develop custom software to execute the migration. The automation will allow the user/modeler to evolve the application without support of the SPO.

Second, **the migration should be performant.** Research on live programming is rapidly gaining traction (see for instance the work of van Rozen and van der Storm [10] and Kubelka et al. [11]). While we do not aim for a live feedback loop, we believe that faster feedback will improve the usability of the MDEE, supporting agile development of the application.

Third, **the migration should not negatively effect the availability of the application.** We aim for a fully automated continuously migration, meaning that scheduling should not be necessary. Migration can thus happen on inconvenient moments (from a business perspective), and therefore should not be noticed by the users. We do not expect users to asses the impact of changes on the availability, and therefore aim to let the migration have no impact at all.

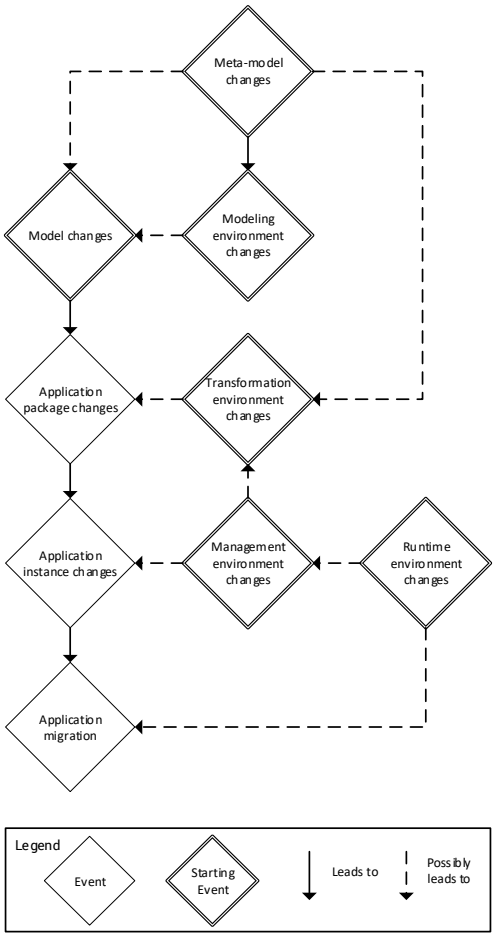

Figure 2. A classification of migration triggers: changes from the MDEE leading to application migration.

Finally, **the migration should be safe**. It should not be possible to deploy an application that does not respond, causes unexpected data-loss, or makes the application non-functional in some other manner.

IV. A Categorization of Migration Triggers

A complete categorization of the migration triggers and the migration strategies is necessary to develop a MDEE that supports continuous migration of application. Migration triggers originate from within the MDEE and require the migration of one or more applications. The triggers are caused by requirements that are implemented by different personas taking action. These requirements are added by the modeler, administrator, or user of the MDEE (see Fig. 1).

Fig. 2 show an initial categorization of the triggers that occur in the MDEE and (possibly) cause an application migration. We recognize that the requirements that cause these triggers (such as changing market requirements or strategic business changes) are important. For instance regulatory requirements lead to market requirements which can lead to model changes (when the regulations are part of the model) or to runtime environment changes (when the regulations influence technical decisions). However, due to scope restrictions these are not discussed at this point. We discuss three of the identified triggers and a possible migration strategy as examples.

The first and maybe most essential trigger is the change of a model by the modeler. The changed model is processed to produce a changed application package, and this application package needs to be redeployed in order to upgrade the application instance. The change of the model could also be triggered by a meta-model change or a change to the modeling environment. In the last two cases, all existing models could be changed instead of a single model.

A second trigger is a change to the transformation environment. Whenever the model transformations are changed, existing models need to be reprocessed to produce updated application packages. Again, those updated packages need to be redeployed in order to upgrade the instances.

The third trigger that we discuss as an example is a change to the runtime environment. When a change to the runtime environment causes changes to the application instance, the management environment also needs to change. Therefore the application needs to be migrated, in fact, all deployed applications need to be migrated.

V. Integrating the Microservice Architecture Style

A second solution approach that we are focusing on is the integration of the microservice architecture style. We believe that this architecture style makes it possible to achieve fine-grained incremental migration. The integration of this architecture style manifests itself in two ways.

First of all, the runtime environment should be able to host distributed microservices. The benefits of this are an improved upgradability, scalability, resilience, and resource sharing. The improved upgradability is the most important: the possibility of updating a single service without affecting the other services. The microservices make it possible to do fine-grained migrations.

While not necessary, we believe that this style needs to percolate all the way through to the meta-model. When the engineers enforce thinking in clear boundaries, smaller independent model elements will arise in the meta-model. These smaller model elements again help to transform the model into microservices.

Second, the transformation environment should consist of independent microservices. This enables incremental transformation of the model through the pattern *Incrementality by traceability* as described by Varró [12].

VI. Data Migration in Event Stores

In earlier work (Overeem et al. [13]) we researched data conversion in event sourcing and proposed a framework to execute these conversions. Event sourcing is a software architecture pattern in which not the current state of an application is stored, but every change leading to the current

state is stored as a log of changes. These events can be used to derive the current state, but they can also show how that state was reached. Common in event sourced systems is the usage of multiple data models. The first data model is the event store, the store with all state changes. It is the most important data model, and recognized as the source of truth for the state. The other data models (which can be as many as required) are derived from these events. Examples are for instance:

- a relational data model with the current state of the application,
- a full text search index, and
- a timeline showing the history of certain objects.

Migration of applications in an MDEE do not only impact the application, but also the application state. The data that is stored by the application might need to be converted to conform to the new desired state. We believe that adopting event sourcing in the runtime environment of the MDEE simplifies data migration. In event sourced systems only the events need to be transformed, because all other data models can be derived from these events. We believe that in many scenarios these events do not require migration, and thus data migration in MDEEs is less complex when using event sourcing.

VII. Related Work

Meijler et al. [14] research fine-grained evolution for generated applications. However, they take the viewpoint of a single application, instead of a multi-tenant MDEE with multiple applications. The integration of the model environment and the runtime environment is seen by Meijler et al. [14] as crucial to achieve fine-grained evolution. The solution they descibe is focused on Model-Driven Architecture (MDA) and the JVM environment. We propose a separate management environment instead of a tight integration of the model and runtime environment. Our approach is less coupled to specific technology or meta-models.

Bruneliere et al. [15] propose Modeling as a Service (MaaS), the synergy of cloud computing and MDE. The research agenda they propose focuses on bringing (parts of) the MDEE to the cloud, such as the modeling environment and the transformation engine. The (continuous) migration of resulting applications is not mentioned. Popoola et al. [16] survey different MDE tools and if they are capable of delivering MaaS functionality. This research too focuses on the modeling and transformation functions of the MDEE.

Scalable MDE is researched by Rajbhoj and Kulkarni [17], Kolovos et al. [18], Cuadrado and de Lara [19]. Rajbhoj and Kulkarni [17] focuses on the scalability of modeling: collaboration and model management. Kolovos et al. [18] define a research agenda for scalable MDE, in which they focus on language design, transformation, collaboration, and persistence. Cuadrado and de Lara [19] specifically research model transformations, and how they can be made streaming.

Our solution proposes integration of the microservice architecture in the MDEE. Sorgalla et al. [20] discuss how microservices can be generated from MDE platforms. Their research group has published more related research: Diepenbrock et al. [21], Rademacher et al. [22], Wizenty et al. [23]. They focus on the team organization and autonomy.

VIII. Conclusion

This paper discusses the research that we are conducting on continuous migration of applications. This challenge is prominent in MDEEs that are used to develop and execute ESAs. The promises offered by MDEEs for ESAs (increased quality, productivity, and flexibility) are only delivered when the challenge of continuous migration is solved. We discussed three solution approaches.

First of all, a categorization of migration triggers including possible migration strategies. This categorization increases the understanding of the challenge. Optimized strategies can be developed when the challenges are clear.

Second, the integration of the microservice architecture style in the MDEE makes fine-grained incremental migration possible. The runtime environment benefits from loosely coupled microservices, because it increases scalability and flexibility. Instead of re-deploying the complete application on every migration trigger, the microservices allow to update only part of the application.

Finally, the adoption of event sourcing in the runtime environment removes complexity from the data migration challenge. The essential characteristic of event sourcing is that every state change is stored as an event. Data models used to present the state of the application are derived from these events. As a result of this approach, these data models are volatile and can be rebuild when needed. These data models thus do not require migration.

These three solution approaches are based on an ongoing case study at AFAS Software. They form the hypotheses that direct our research. These three solution approaches tackle the challenge of continuous migration from different angles.

References

[1] V. G. Díaz, E. R. N. Valdez, J. P. Espada, b. C. P. G. Bustelo, J. M. C. Lovelle, and C. E. M. Marín, "A brief introduction to model-driven engineering," *Tecnura*, vol. 18, no. 40, pp. 127–142, 2014.

[2] J.-P. Tolvanen and S. Kelly, "Model-Driven Development Challenges and Solutions - Experiences with Domain-Specific Modelling in Industry," *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development*, no. January, pp. 711–719, 2016. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0005833207110719

[3] R. F. Paige and D. Varró, "Lessons learned from building model-driven development tools," *Software & Systems Modeling*, vol. 11, no. 4, pp. 527–539, 2012. [Online]. Available: http://link.springer.com/10.1007/s10270-012-0257-9

[4] T. Clark and P. A. Muller, "Exploiting model driven technology: A tale of two startups," *Software and Systems Modeling*, vol. 11, no. 4, pp. 481–493, 2012.

[5] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

[6] Gartner, "Enterprise Application Software," 2012. [Online]. Available: https://www.gartner.com/it-glossary/enterprise-application-software

[7] A. W. Brown, "An introduction to Model Driven Architecture," *The Rational Edge*, pp. 1–16, 2004. [Online]. Available: http://www.ibm.com/developerworks/rational/library/3100.html

[8] C. Krueger, "Easing the Transition to Software Mass Customization," in *International Workshop on Software Product-Family Engineering*. Springer, 2002, pp. 282–293. [Online]. Available: http://link.springer.com/10.1007/3-540-47833-7{_}25

[9] M. Overeem, S. Jansen, and S. Fortuin, "Generative versus interpretive model-driven development: Moving past 'It depends'," in *Communications in Computer and Information Science*, vol. 880, 2018, pp. 222–246.

[10] R. van Rozen and T. van der Storm, "Toward live domain-specific languages: From text differencing to adapting models at run time," *Software and Systems Modeling*, pp. 1–18, 2017.

[11] J. Kubelka, R. Robbes, and A. Bergel, "The road to live programming," *Proceedings of the 40th International Conference on Software Engineering - ICSE '18*, pp. 1090–1101, 2018. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3180155.3180200

[12] D. Varró, "Patterns and styles for incremental model transformations," in *CEUR Workshop Proceedings*, vol. 1657, 2016, pp. 41–43.

[13] M. Overeem, M. Spoor, and S. Jansen, "The Dark Side of Event Sourcing: Managing Data Conversion," in *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017, pp. 193–204.

[14] T. D. Meijler, J. P. Nytun, A. Prinz, and H. Wortmann, "Supporting fine-grained generative model-driven evolution," *Software & Systems Modeling*, vol. 9, no. 3, pp. 403–424, 2010.

[15] H. Bruneliere, J. Cabot, and F. Jouault, "Combining Model-Driven Engineering and Cloud Computing," in *Modeling, Design, and Analysis for the Service Cloud-MDA4ServiceCloud'10: Workshop's 4th edition (co-located with the 6th European Conference on Modelling Foundations and Applications-ECMFA 2010)*, 2010.

[16] S. Popoola, J. Carver, and J. Gray, "Modeling as a service: A survey of existing tools," *CEUR Workshop Proceedings*, vol. 2019, pp. 360–367, 2017.

[17] A. Rajbhoj and V. Kulkarni, "Large scale model-driven engineering for a multi-site team-Experience report," *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, no. 2, pp. 123–128, 2013.

[18] D. S. Kolovos, M. Tisi, J. Cabot, L. M. Rose, N. Matragkas, R. F. Paige, E. Guerra, J. S. Cuadrado, J. De Lara, I. Ráth, and D. Varró, "A research roadmap towards achieving scalability in model driven engineering," *Proceedings of the Workshop on Scalability in Model Driven Engineering - BigMDE '13*, pp. 1–10, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2487766.2487768

[19] J. S. Cuadrado and J. de Lara, "Streaming model transformations: Scenarios, challenges and initial solutions," in *International Conference on Theory and Practice of Model Transformations*. Springer, 2013, pp. 1—-16.

[20] J. Sorgalla, F. Rademacher, S. Sachweh, and A. Zündorf, "On Collaborative Model-driven Development of Microservices," in *MSE Workshop @ STAF2018*, 2018, pp. 1–8. [Online]. Available: http://arxiv.org/abs/1805.01176

[21] A. Diepenbrock, F. Rademacher, and S. Sachweh, "An Ontology-based Approach for Domain-driven Design of Microservice Architectures," *INFORMATIK 2017*, no. September, pp. 1–12, 2017.

[22] F. Rademacher, J. Sorgalla, and S. Sachweh, "Challenges of Domain-Driven Microservice Design," *IEEE Software*, p. 8, 2018.

[23] P. Wizenty, J. Sorgalla, F. Rademacher, and S. Sachweh, "Magma: Build Management-based Generation of Microservice Infrastructures," *Proceedings of the 11th European Conference on Software Architecture Companion Proceedings - ECSA '17*, pp. 61–65, 2017. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3129790.3129821