Model-driven Engineering Tool Comparison for Architectures within Heterogenic Systems for Electric Vehicle

Sebastian Apel, Marianne Mauch and Volkmar Schau
Department of Computer Science, Friedrich Schiller University Jena, Ernst-Abbe-Platz 2, 07743, Jena, Germany

Keywords: Software Architecture, Code Generator, Model-driven Engineering, Electric Vehicles, Logistic, Freight Transportation.

Abstract: System landscapes within logistical scenarios is highly heterogenic. Adding specific mechanisms, e.g. to support planing, monitoring and analyses for fully electrical powered vehicles, could become a mess or at least a challenge. While our project Smart City Logistic (SCL) is trying to manage this extension for multiple logistic scenarios, other projects want to do comparable system extensions as well. The following approach tries to evaluate how model-driven engineering (MDE) combined with generative frameworks can support the transfer from platform independent models to deployable solutions within the logistical domain. This paper compares specific MDE tools which can be used to support such a framework.

1 INTRODUCTION

Enabling inner-city logistics with fully electrical powered vehicles is a challenge. Exploring the reasons behind this challange will lead to infrastructural issues, like missing charging stations, but also to missing trust in predicted ranges. The German Federal Ministry for Economic Affairs and Energy (BMWi) initiated the collaborative research project SCL to support this field of application. So the project combines an intelligent route planning system with dynamic range prediction, based on the detailed knowledge of major influencing factors. The SCL information and communication technology (ICT) system is able to support dispatchers and drivers with real-time feedback, optimizing the driving style and providing alternatives for successful ends of planned tours.

These goals will be achieved by using three main components: (1) a mobile assistance client for drivers to display real-time information with offline capabilities, (2) a hardware telematic unit, connected with the Electric vehicle (EV) on-board electronic, collecting and broadcasting required data as well as (3) a centralized server to manage, analyse and persist information about transmitted data and offer planned tours. But the challenge of those kind of systems, to enable EVs in specific scenarios, is to embed them in highly closed logistic system landscapes.

This model-driven architecture (MDA) tool comparison will proof, how model-driven engineering and generic infrastructures can be used for reduction of development effort by using already created platform-independent model (PIM). SCL wants to use the results and knowledge about MDE tools within a generative framework for the SCL reference architecture in ICT-systems which enables to extend existing logistical landscapes.

This paper starts with a SCL requirement engineering to show a list of needs for this EV-domain by analysing the logistical partners. The results are used to create an architectural draft, the PIM, which shows the main entities, components and concepts. Based on this minimal subset of elements, used by this PIM, simplified test cases can be re-engineered. These entities will be used within multiple model-driven engineering tools to generate source code and to gain an insight into their generative capabilities.

2 REQUIREMENTS

The SCL ICT-system supports the usage of EVs by combining components linked over different types of communication technologies. Figure 1 gives a first insight and visualizes already named main components like the mobile assistance client, the telematic unit and the centralized server in addition to other components like the electronic freight monitoring and the existing ICT-systems which SCL has to use as data sources and sinks.

Apel, S., Mauch, M. and Schau, V.
Model-driven Engineering Tool Comparison for Architectures within Heterogenic Systems for Electric Vehicle.
DOI: 10.5220/0005803806710676
In *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, pages 671-676
ISBN: 978-989-758-168-7

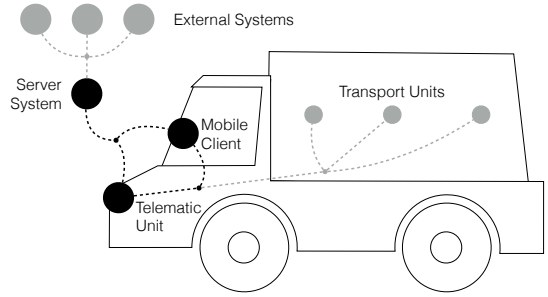


Figure 1: Simple IT-system overview of SCL.

First step was to analyse business processes in inner-city logistic companies. These analyses were done by interviewing actors like dispatchers and drivers about their everyday work (Apel et al., 2014). Which tools and software are used? Which workflow will be handled and which exceptions could occur? As a result of that multiple business processes about planning tours, handling orders, managing picking and packing and executing delivery were created. These processes could be compared between different companies, but they are not identical at all. Since order handling as well as picking and packing mostly managed by the ICT-systems, the tour planning and delivery processes is the important point to inject additional services to handle range restrictions of EVs. The following list of basic use cases shows necessary elements for an extension system like SCL that has to create.

- Collection of core and runtime data, like driver, car, customer and order data
- Configurable interfaces to handle the wide range of connectivities of transport management system (TMS) and warehouse management system (WMS) solutions
- Calculate a forecast of energy consumption when using EVs in a specified situation, energy optimized routes by using calculation forecasts and optimized tours by using energy optimized routes
- Planning, monitoring, analyses and optimizing tours by using core and runtime data
- A knowledge base about data and dependencies used by this system
- Interfaces to gather data about weather, traffic and vital data from EVs
- Managing exceptions in case of divergences

Followed by the interviews next step was to take a closer look at software used by logistical companies. This software was found as a subset of enterprise resource planning (ERP) software, especially the supply chain management (SCM) systems (Chopra and Meindl, 2007). In case of inner-city logistics this would be a TMS, a WMS or an intersection of their functionalities. By analysing available interfaces (Busch et al., 2003; Logistik Heute, 2005; Pirron et al., 1999; Faber and Ammerschuber, 2007) to extract order related details for tour calculations, it is possible to find something like HTTP based interfaces (for example Representational State Transfer (REST) or Simple Object Access Protocol (SOAP)) with data representation in XML or JSON, interfaces defined by leading companies (for example SAP), interfaces for financial information like DATEV and financial accounting, file based interfaces like spreadsheets, ASCII or XML files, as well as interfaces based standardized content based on electronic data interchange (EDI) with transportation layers based on HTTP or File Transfer Protocol (FTP) for example.

Analysing those TMS and WMS software would not lead to commonly accepted standards for covering a dominant set of interfaces. Each solution presents their own interfaces. Some of them uses comparable types of interfaces (like REST), other one export functionalities based on EDI, but unfortunately no common standard at all.

Rising numbers of innovations and technologies requires more flexibility of these closed ICT-systems. Projects like Econnect and sMobility point out how charging management can save the electricity infrastructure. Using EVs as energy buffers to keep remaining energy and pass back when requested is shown in projects like iZEUS by SAP. Finally SCL and iZEUS shows how tour and order related data can be used to optimize available ranges when using EVs. Waiting until each TMS and WMS development company embed each specific use case into their own solutions the challenging launch of EVs would slow down massively.

3 ARCHITECTURE

The server architecture has to regard the SCLrequirements and orchestrate related components to handle tasks about planing, monitoring and analyses of tours. Additionally, this orchestration has to divide those tasks semantically in components used as interfaces, views, managing or controlling, as well as a model.

A look into related work of logistic based architectures shows several approaches improving them or specifying the content and meaning of communications between different companies. These approaches describe single systems, architectures or object models to manage situations within companies (Re et al., ;

Moore et al., 1996; Satapathy et al., 1998; C.Quiroga et al., 2011). In case of adding additional services, to handle new innovations and offer a wider field of business models, a common procedure would be proposed to extend or replace partly the existing system. Neither SCL nor the related logistical partners don't want to replace their systems just to use EVs. There is a urgent need of a supporting system which can be used in addition to existing ICT-system landscapes.

SCL has to create an own server architecture to handle the requirements about a system which extends existing ICT-systems. As a basic semantic structure the server uses the Model-View-Controller (MVC) pattern (Buschmann et al., 1996). Combining those ideas with loose dependencies, which is required to easily extend existing systems, a specific communication between their MVC components is required. The use of ideas of micro kernel patterns (Buschmann et al., 1996) as well as mechanisms within observer patterns (Gamma et al., 1994) in combination with an enterprise service bus (Chappell, 2004) leads to an architecture as shown in figure 2. Each component is a specialized micro service for small subsets of tasks. Components can broadcast events to everybody as well as starting a private communication based on a reply to previously received message. The common workflow for data exchange would perform a broadcast about some specific event with public available information. An interested component listens for events related to those specific type of information and can reply to this in order to ask for more information. The original event broadcaster would receive this reply and can decide to answer with more detailed, mostly private information.

The architecture in figure 2, which combines the described communication and MVC semantic, defines four components used to handle model specific tasks. The first one describes a precise range prediction model (3). The input would be one or multiple vectors of parameters which describes e.g. a road segment based on height, speed, time and weight. The range model will use these vectors to calculate a predicted consumption based on previously learned situations. This range component can be utilized by using modified route calculations (4) getting energy optimized routes, which can be embedded within a third component to calculate optimized tours (5). In addition, all related data about drivers, EVs, orders and shipments as well as charging stations, users, companies and customers would be handled within a knowledge base.

Beside model components, a bunch of components to handle different types of requests has to be added (2). Those components handle requests about

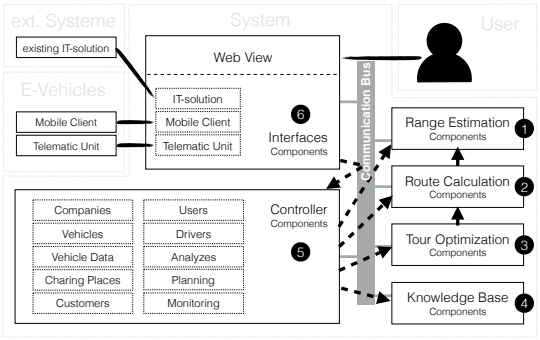


Figure 2: Architectural overview.

tasks like monitoring, planning and analyses, as well as managing their related core data like available EVs, drivers, users and customers in addition to basic data like EV type data and charging places.

The last part about interfaces and views would be the entry point for external systems into this server architecture. Because of this wide range of in section 2 mentioned possible interfaces, data formats and protocols SCL has to use a configurable interface system which can be easily modified to match specific requirements. In preparation three strategies were developed to handle communication in SCL. The first one handles this wide range by creating a inheritance tree for different specializations of interfaces. The second one use the pipeline concept (Buschmann et al., 1996, p. 53) to split each transformation step within a communication stack defined by the OSI reference model (ISO/IEC JTC 1, 1994). Each step like TCP, HTTP, Transport Layer Security (TLS) can be rearranged in a fixed order to handle a specific communication stack like shown in figure 3. The last one describes a strategy where the interface itself isn't handle directly. The architecture should describe a well defined application programming interface (API) for each existing request type. An interface developer has to use this API and implement the whole interface as a plugin for this system. Because of the expanding number of dependencies in case of using of inheritance trees and heavily not reusable characteristics of an API strategy, the SCL platform uses the pipeline strategy to get configurable interfaces.

MVC, internal communication, data model defined by the knowledge base, micro services described within components and the idea about configurable

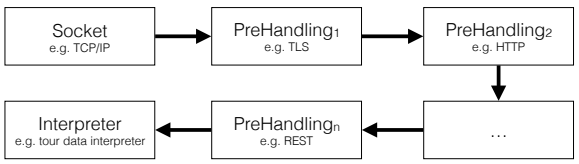
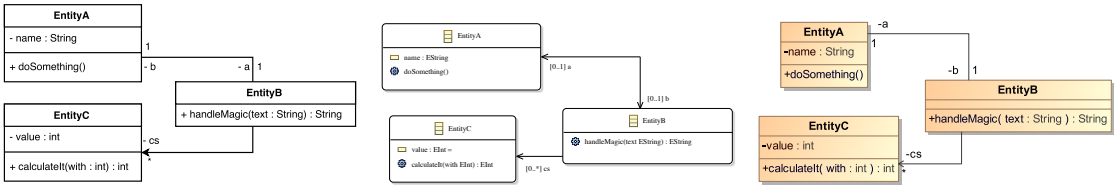


Figure 3: Pipeline strategy for interface components.



(a) (b) (c)

Figure 4: Simple test model with three entities (a) mapped to EMF (b) and MagicDraw (c).

interfaces based on pipelines would be the SCL PIM. Apart from its manual realization within this project, SCL proves what can be generated by using MDE tools.

4 MDE TOOLS IN PRACTISE

SCL wants to use the knowledge about the engineering of embeddable logistical extension systems to refactor a generic service framework. Based on this requested framework, it raises the question about which parts could be covered with generic processes.

Using everything related from the PIM about components and interfaces would lead to a whole bunch of entities, too much to get through available generator tools. To simplify this model for evaluating tools an abstracted test model as shown in figure 4(a) has to be used to reduce complexity, but reflects specific elements from SCL. The model contains classical attributes, single- and bidirectional associations and methods with basic attributes and return values.

4.1 Overview

There are several tools supporting the model-driven engineering concept. A lot of related work about how to deal with MDA subject, available tools and case studies were published. Beydeda (2005) summarize projects employing MDE and Rempp (2011) introduces modeling methods, languages and tools which can be used as foundation for the SCL MDA tool comparison. Silingas (2007) describes a conceptual framework for model-driven development (MDD) combined with “a case study of modeling software for library management”. Calic (2008) on the other hand works out criteria about an ideal MDA tool for exemplifying the development of an architecture for a glossary management tool. Finally Alford (2013) and Silva (2010) compare different modeling tools which can be used as a initial listing. Within this related work tools like MagicDraw, EMF, AndroMDA, BlueAge Forwarded, Cameo E2E Bridge, Mia Software, Jamda, PathMATE, Rapsody, Modelio and tools from Sparxsystem can be found.

Table 1: List of considered tools.

Name	Dependency	Runtime Environment (e.g.)	Licensing	Generator Results
MagicDraw	/	plain code like C++ or Java	Commercial	Generative Artifcats
EMF	/	plain code like Java, extendable with plugins	Free	Generative Artifcats
AndroMDA	MagicDraw, EMF or other	Java EE / .NET	Free	Generative Artifcats
BlueAge Forwarded	MagicDraw or EMF	Java EE / .NET	Commercial	Full, by using activity diagrams
Cameo E2E Bridge	MagicDraw	Cameo E2E Bridge Server	Commercial	Full, by using BPMN

Some tools, like Jamda, were skipped because of their outdated maintenance. Other tools, like PathMATE and Mia Software, were ignored because of missing access through their tools. Furthermore, some interesting tools like Rapsody from IBM, Modelio from Modeliosoft or tools from Sparxsystem were skipped in respect to project time scopes. The chosen tools in table 1 scopes an important known subset of available solutions.

Based on the specific SCL requirements about creating a generative framework in addition to the SCL reference architecture, a classification of available tools has to be done. As shown in table 1, these classifications were done based on tool dependencies, the corresponding runtime environment for generative results, the offered licensing models and about their generative capabilities (“Generator Results”).

Within the scope of SCL and based on the requirement to have different kinds of generative results, three tools are selected. EMF is selected because of the internal description mechanisms. MagicDraw on the other hand were chosen because of the extended modeling features within the Unified Modeling Language (UML) universe. In addition to MagicDraw and EMF AndroMDA were chosen because of the capabilities to use MagicDraw and EMF Models as sources to work with an specific runtime environment as well as the way they offer mechanisms to extend their tool. Cameo E2E Bridge and BlueAge Forwarded get dropped because of their required runtime environment.

4.2 Eclipse Modeling Framework

EMF as a powerful framework, embedded within the Eclipse IDE, can be used as a code-generation facility for building Java applications. The model created in EMF works as a connection between XML structures, Java code and UML diagrams. Within EMF Ecore is used to represent a meta model to describe those models, like attributes, classes and references, and can be used to describe each part of the domain specific elements. This is were the basic data types comes from, e.g. integers are known as EInt, strings are known as EString and so on. (Budinsky, 2004)

A created class diagram based on figure 4(a) would result in something like figure 4(b). This is nearly the same as the original diagram with slightly modified attribute types to match the meta model from Ecore. After creating this EMF model by using class diagrams, it is possible to create a generator model. Generators can be used to map this EMF model to Java code as well as XML schemata. Looking through generated Java code some differences between intended entities within the class diagram and the generated result will be shown. Each class (EntityA, EntityB and EntityC) will be mapped through an interface. The implementation, along attributes, can be found in classes which implements those interfaces (e.g. EntityAImpl). In addition to interfaces and implementations, a couple of patterns and features can be realized like factory pattern, test packages, instance editors and meta data about entities (e.g.).

4.3 MagicDraw

MagicDraw from NoMagic is an UML tool suite, which covers most parts like class, sequence, activity, state and component diagrams. Furthermore this tool suite has embedded generative capabilities for bidirectional mapping between entities and source code. The model is managed within the MagicDraws own data format and can be exported through Ecore models to use them within EMF.

Using MagicDraw to create a class diagram from figure 4(a) would result in something like figure 4(c), which might be exactly the same as the intended one. MagicDraw let you choose the required target language by creating a specific generator. In case of using Java as target language, MagicDraw generates, nearly out of the box, the whole entity including method signatures and attributes. Adding stereotypes allows MagicDraw to add functional elements like implementations for getters and setters. In general this generated code will be written as modelled. Classes will be mapped through classes, interfaces will be mapped through interfaces and generalizations are used as intended.

4.4 AndroMDA

AndroMDA is a modularly configured and open source MDA generator framework and is oriented towards Object Management Group (OMG) MDA standard. By comparison to MagicDraw or EMF AndroMDA doesn't work as a modeling tool itself. This framework refers to external tools, in detail to their external metamodels e.g. MagicDraw or Ecore from EMF. The central element of AndroMDA are cartridges. AndroMDA supports BPM4Struts, jBPM, JSF, EJB, C++, Java and because of its modular built structure it is extendable by adding further cartridges.

Using AndroMDA will start with several questions and guides developers to get their first starter application. Questions about modeling tools, project name, id, version and detailed information about type of application to generate (e.g. rich client, j2ee), type of packaging (e.g. EAR or WAR), type of transactional / persistence cartridge (e.g. hibernate, ejb, ejb3, spring, none), type of database backend for persistence layers, type of workflow engine capabilities, as well as web user interface. Sadly the choice about which kind of application, furthermore the target runtime environment and dependencies, has to be done quite before modeling starts. AndroMDA can easily be used within a platform-specific model (PSM) phase, when modeling has to be created from scratch.

5 CONCLUSIONS

MDE is challenging in specified use cases with predefined runtime environments. This is getting much more difficult in case of generative frameworks which supports development processes based on a platform independent reference architectures. The tested tools shows that it is quite easy to map entities bidirectional between model and source code. Unfortunately the modeling of processes or procedures – e.g. with BPMN or activity diagrams – highly depends to specific development and execution environments. Till now those environment decision has to be done in the beginning of the modeling phase, which can not be done in case of platform independent modeling and reference architectures. This decision has to be done by the respective developer.

However, the available tools and meta descriptions can be used within the intended framework. Ecore models seem to be a commonly accepted metalanguage for entities which can be edited by a lot of free

and commercial tools like EMF and MagicDraw. In addition to that it is easy to get generated code from those models which are based on Ecore.

The missing possibility to describe processes or procedures has to be added to get the SCL framework as intended. But furthermore there are tools which could be used as base to prevent development from scratch. AndroMDA shows basic structures to extend the generative capabilities by adding new cartridges which could extend the plain entity to code mapping with specific behavior.

SCL will use the results to go further through a generative framework to support development by using the SCL reference architecture for logistical system extensions. The next step would be to define and describe how internal mechanisms can be modeled without losing platform independences.

ACKNOWLEDGEMENTS

We would like to thank all members of the SCL research team here at FSU Jena – there are too many to name them all in person. We would also like to extend our gratitude to our partners within the research consortium, end users as well as research institutions and industrial developers. This project is supported by the German Federal Ministry for Economic Affairs and Energy in the IKT-II fur Elektromobilitat program under grant 01ME121(-33).

REFERENCES

Alford, R., Simmonds, D., Reinicke, B., and Vetter, R. (2013). An evaluation of model driven architecture (mda) tools. Master's thesis, Annals of the Master of Science in Computer Science and Information Systems at UNC Wilmington. 7(2) paper 5.

Apel, S., Schau, V., and Rossak, W. (2014). Darstellung branchentypischer Abläufe und Wechselwirkungen in der innerstädtischen logistik. Technical Report Math/Inf/06/2015, Friedrich-Schiller- Universität Jena, Institut für Informatik, Jena, Germany.

Beydeda, S., Book, M., and Gruhn, V. (2005). *Model-Driven Software Development*. Springer Berlin Heidelberg.

Budinsky, F. (2004). *Eclipse Modeling Framework: A Developer's Guide*. The eclipse series. Addison-Wesley.

Busch, A., Dangelmair, W., Pape, U., and Rüther, M. (2003). *Marktspiegel Supply Chain Management Systeme*. Gabler, Wiesbaden.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. (1996). *A System of Patterns, Pattern-Oriented Software Architecture*. John Wiley and Sons.

Calic, T., of Nevada Reno, U., Dascalu, S., and Egbert, D. (2008). Tools for mda software development: Evaluation criteria and set of desirable features. In Latifi, S., editor, *Proceedings of the Fifth International Conference on Information Technology: New Generations*, pages 44–50. IEEE Computer Society. ITNG 2008.

Chappell, D. (2004). *Enterprise Service Bus*. O'Reilly Series. O'Reilly Media, Incorporated.

Chopra, S. and Meindl, P. (2007). *Supply Chain Management*, volume 3. Pearson Education, New Jersey.

C.Quiroga, N.Koncz, E.Kraus, J.Villa, J.Warner, Y.Li, and D.Winterich (2011). Guidance for developing a freight transportation data architecture. *NCFRP Report 9*.

Faber, A. and Ammerschuber, O. (2007). Transporte im Griff. *Logistik Heute*, pages 35–37.

Gamma, E., R.Helm, Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 edition.

ISO/IEC JTC 1 (1994). *ISO 7498-1: Information technology - Open Systems Interconnection - Basic Reference Model*, 1 edition.

Logistik Heute (2005). Die Qual der Wahl. *Logistik Heute*, pages 36–37.

Moore, M., Kumara, S., and Richbourg, R. (1996). An architecture for logistics replanning. *Expert Systems with Applications*, 11(2):177–190.

Pirron, J., Kulow, B., Hellingrath, B., and Laakmann, F. (1999). Marktübersicht SCM-Software. *Logistik Heute*, pages 69–75.

Re, S. D., Campagna, A., and Nanni, U. A reference architecture for freight transport manangement systems. Internet, http://freightwise.tec-hh.net/A-reference-architecture-for-freight-transport-management-systems.pdf, visited 2015-10-21.

Rempp, G., Akermann, M., Löffler, M., and Lehmann, J. (2011). *Model Driven SOA, Anwendungsorientierte Methodik und Vorgehen in der Praxis*. Xpert.press. Springer-Verlag Berlin Heidelberg.

Satapathy, G., Kumara, S., and Moore, L. (1998). Distributed intelligent architecture for logistics (dial). *Expert Systems with Applications*, 14:409–424.

Silingas, D. and Vitiutinas, R. (2008). Towards uml-intensive framework for model-driven development. In Meyer, B., Nawrocki, J., and Walter, B., editors, *Balancing Agility and Formalism in Software Engineering*, volume 5082 2008 of *Lecture Notes in Computer Science*, pages 116–128. Springer Berlin Heidelberg.

Silva, J. and Arévalo, D. (2010). Modelo comparativo de herramientas mda en ambientes de software libre. Technical report, Universidad Autónoma de Bucaramanga, Colombia. Internet, http://de.scribd.com/doc/34353208/tecnico-Jamda, visited 2015-10-21.