BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 58, No. 3, 2010
DOI: 10.2478/v10175-010-0039-x

Model-driven systems engineering: state-of-the-art and research challenges

M. LAUDER[1]*, M. SCHLERETH[2], S. ROSE[1], and A. SCHÜRR[1]

[1] Real-Time Systems Lab, Technische Universität Darmstadt, Merckstr. 25, 64283 Darmstadt, Germany

[2] Siemens AG, Gleiwitzer Str. 555, 90475 Nuremberg, Germany

Abstract. Model-driven software engineering is a well investigated and heavily used technique for software development. Within automation engineering we want to benefit from these ideas and concepts by adopting them to systems engineering. Parallel processes in systems engineering demand high synchronization effort between different disciplines, their engineers, and processes. Unfortunately, these processes are concurrently established, but do only support sequential engineering. With model-driven systems engineering we want to enable systems engineers to model their domain knowledge and tooling on a more abstract level. Thus, engineers may benefit in increasing efficiency and quality for the resulting products from existing integration approaches of engineering artifacts and tools. By means of an integration approach we are able to synchronize and check consistency of model data that evolved parallel in different tools. In this contribution, we present a new classification scheme for integration scenarios and explain our modeling and integration approach together with a proof-of-concept use case and prototype, located in automation engineering.

Key words: model-driven development, integration, automation engineering, TGG, empirical.

1. Introduction

The development of automation systems for machines and plants depends on information from an increasing number of engineering tools like mechanical CAD, automation device configuration or control logic engineering. Automation engineering of programmable logic controllers (PLC) requires for example information about the devices used for machine automation, their characteristics, and their interaction with other machine modules from other engineering tools like electrical engineering or mechanical engineering.

Since information exchange between these tools and design models is mostly based on design documents and meetings, there is a strong requirement for a tighter integration of PLC engineering models to raise design efficiency. Automation system providers like machine builders drive the integration of PLC engineering by the establishment of mechatronic development processes which shall integrate mechanical engineering, electrical engineering and automation engineering [1]. Automation system users like automotive companies work at the realization of the digital factory [2] with engineering models available for design, commissioning and operation of production sites.

Concurrent Model Driven Automation Engineering (CMDAE) addresses the requirement for the integration of PLC engineering with other disciplines and establishes a bidirectional data synchronization between the design models used for the development of automation systems. In this paper, the term PLC refers to both hardware and software technology and not only to a special hardware architecture (e.g. in contrast to PC-based soft logic controllers). The software technology of PLC controllers is defined by the standard IEC 61131 [3].

This paper reports on the current status and findings of our prototype implementation for the integration of the location-oriented machine structure defined in electrical engineering with the hardware configuration of a PLC defined in Simatic Step7. Section 2 describes the application scenario that we used to understand the requirements for the model-driven tool integration. The engineering workflow behind the engineering tool integration is described with respect to the data synchronization between electrical engineering and automation engineering. Section 3 presents the CMDAE classification scheme, which helps tool integration developers in the communication about the requirements and domain models of the tool users. Based on the positioning of the integration task in the CMDAE classification scheme, the integration developer can select the adequate integration technology. The implementation technologies used by CMDAE – metamodeling and triple graph grammar – in combination with the implementation of the integration of the electrical engineering tool Comos ET and the automation engineering tool Simatic Step7 are introduced in Sec. 4. Section 5 demonstrates the application of the tool integration environment developed in Sec. 4 according to the example workflow in Sec. 2. Finally, Sec. 6 and 7 conclude this paper with references to related work, summary and open challenges.

2. Application scenario: integration of electrical engineering and PLC engineering

Machine development is based on an established development process of machine builders in an existing tool environment. Therefore, the introduction of an environment with

*e-mail: marius.lauder@es.tu-darmstadt.de

model exchange between these engineering tools is usually an a-posteriori integration of existing tools and tool interfaces.

Our application scenario demonstrates the workflow between electrical engineering (based on Comos ET [4]) and automation engineering (based on Simatic Step7 [5]), the application of concurrent model-driven engineering (CMDAE). As an automation application example, we look at the automation of a storage and retrieval machine of a high-bay warehouse system. The storage and retrieval machine runs within a warehouse aisle (see Fig. 1 right-hand side) and picks or places goods from the storage shelve.

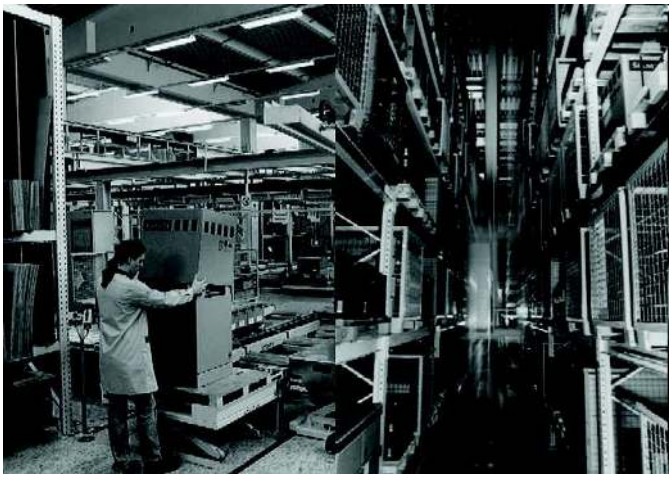

Fig. 1. High-bay warehouse application example

The overview of automation devices in Fig. 2 shows the automation devices used in this high-bay warehouse system example: a Siemens Simatic CPU 317T-2 DP controller with integrated motion control functions, distributed I/O (input/output) modules with Siemens Simatic ET 200S and motor control by a Siemens Sinamics drive system. Complex components like data matrix systems (e.g. for identification of goods at the commissioning, Fig. 1 left-hand side) or handling robots are connected by fieldbus communication.

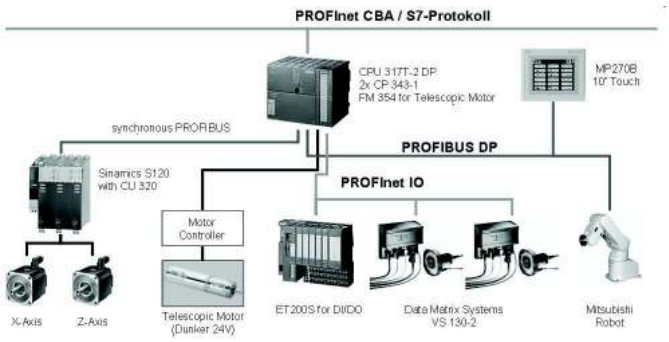

Fig. 2. Example of the automation structure of a storage and retrieval machine

2.1. Integration requirements. In the development process of such a high-bay warehouse system, the configuration of the Simatic I/O modules, shown in the automation structure in Fig. 2, changes if an electrical engineer changes the wiring of the devices built in electrical cabinets. In the opposite direction, changes of I/O modules due to automation programming requirements must be reflected within the electrical engineering tool. This engineering workflow is a basic use case of CMDAE: Propagate changes from an engineering discipline like electrical engineering to another engineering discipline like automation engineering. Usually this is an incremental propagation of some changed elements between two existing models as describe above for Simatic I/O modules. In the special case of a completely new project started in one discipline or code generation scenarios, the model of the other discipline can be created from scratch.

Depending on the responsibilities in an engineering organization, automatic change propagation might not be desired. Instead notifications about inconsistencies between the engineering models should be generated. The resolution of these inconsistencies remains the responsibility of the engineer of each discipline. The implementation of consistency checks is usually easier than change propagation, since the actions required for reconciliation need no implementation.

Both change propagation and consistency checks use traceability links between related elements of different engineering models internally in the CMDAE implementation. Besides the usage for change propagation, traceability is also an important use case for automation engineers to follow related elements, e.g. to find out the machine function a specific I/O module is used for.

As a non-functional requirement the tool integration environment with the features described above must integrate the existing engineering tools of machine builders and not presume the introduction of new engineering tools. Usually machine builders have an existing environment of tools for mechanical, electrical and automation engineering as described in Sec. 1 but with manual data exchange between these engineering tools or hand-crafted uni-directional batch data exchange solutions. Therefore, a tool integration environment must adapt to the existing engineering tools and not vice versa since most engineering tool providers will not adapt their tools to a common integration environment.

The target users of model-driven automation engineering are software engineers working at the IT department of the machine builder, or at a service provider, as well as the engineers working in the engineering disciplines of machine development like electrical engineering or automation engineering. Model-driven automation engineering helps the software engineers with the development of the integration environment and the domain engineers with the communication about the definition of the engineering models and with the implementation of data synchronization workflows between engineering tools. Model-driven automation engineering helps both kinds of users to keep control of the data workflow between integrated engineering tools for specific machine development projects.

2.2. Change propagation workflow example. The detailed integration workflow used in our application scenario is shown in Fig. 3. The initial location-oriented structure of a new ma-

chine is created in the electrical engineering tool Comos ET. For simplification, this location-oriented structure consists in our example of a single PLC with I/O modules. The location-oriented structure is propagated by the CMDAE tool integration environment to Simatic Step7 (Step 1 in Fig. 3). For this propagation step the model of the location-oriented structure is the *source model* of the data propagation. The data propagation is also referred to as a *translation*, since information in the source model has to be translated into adequate information of the *target model*, the Step7 engineering project for this data propagation. The source model is in the sense of a translation the sender of information, the target model is the receiver of information. Since no Step7 project exists yet, an initial Step7 project is created with the single PLC visible in the hardware configuration. With that project at hand, the automation engineer starts developing the PLC program. In our application scenario, the automation engineer realizes that he requires an additional I/O module due to some specifics of the drives used. Therefore, he adds this additional I/O module to the hardware configuration of Step7 (Step 2 in Fig. 3). The next time the electrical engineer decides to update the ECAD project in Comos ET with information from automation engineering, the CMDAE tool integration environment propagates the I/O module added by the automation engineer to the ECAD project (Step 3 in Fig. 3). In this data propagation, the roles of source and target model have changed: the automation engineering model is the source model of the data propagation, the model of the location-oriented structure is the target model.

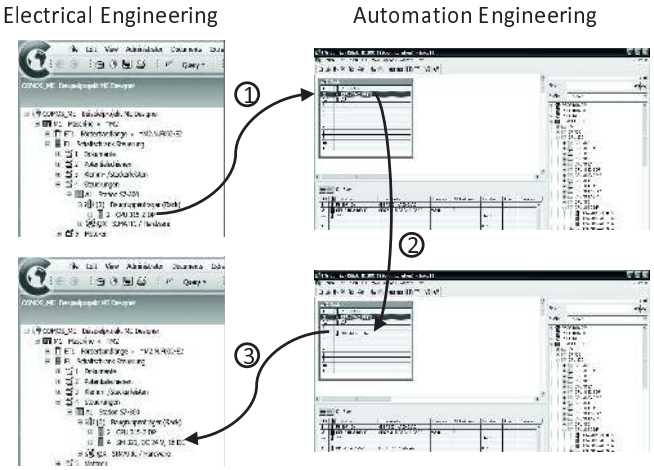

Fig. 3. Integration workflow between Comos ET (left) and Step7 (right)

This engineering workflow is used in the following to demonstrate the CMDAE concepts.

3. Classification scheme

The a-posteriori integration of engineering tools as described in Sec. 2 is a software engineering task that consists of the analysis and formal description of the engineering models used and the implementation of the data exchange based on existing tool interfaces and data repositories. For the classification of these software engineering artifacts and the related development processes, we developed the CMDAE (concurrent model-driven automation engineering) hypercube shown in Fig. 4. The CMDAE hypercube is used for the conceptual description of the models and information used for integration, "their relationships to each other, and to the environment, and the principles guiding its design and evolution" [6].

The CMDAE hypercube consists of the following five dimensions:

- Concurrent engineering disciplines (C) define the relationship of elements in different engineering disciplines.
- Metamodeling (M) is used to define a metadata architecture for the models used in the tool integration scenario.
- Domain Customization (D) enables company and user specific adaptation of the integration environment.
- Abstraction (A) supports modularization, reuse, and support of different implementation platforms within the integration model.
- Evolution (E) covers the administration of different versions of integration artifacts, which change with the evolution of metamodels and engineering tool releases in business organizations.

The five dimensions of the CMDAE hypercube are visualized by a star chart in Fig. 4. Each dimension represents a set of exclusive characteristics for that dimension, e.g. the different engineering disciplines for the dimension C (Concurrent Engineering). A software engineer can use this star chart to analyze and compare different engineering tools. Figure 4 shows the analysis of the integration of the ECAD tool (dashed line) and of Simatic Step7 (dotted line). It is noticeable in the star chart that the tools in our scenario share the same characteristics on each dimension of the CMDAE hypercube except for the dimension C, which is the integration of the engineering disciplines electrical engineering and automation engineering. Therefore, the implementation of the integration environment can focus on that aspect and, for example, needs not to deal with different versions of the metamodels involved in dimension E.

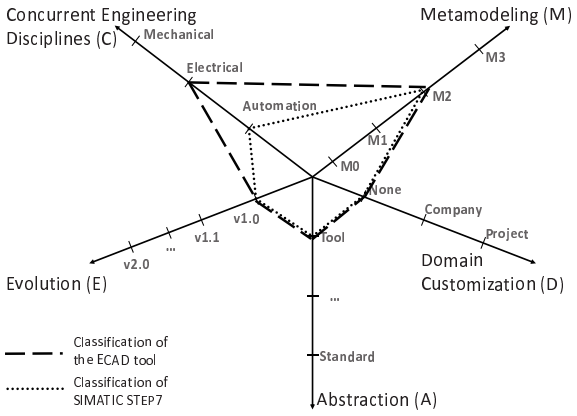

Fig. 4. Classification of the application scenario within the CMDAE hypercube

In the following, we will use the integration scenario for the engineering of the automation system of the high-bay warehouse system, introduced in Sec. 2, to explore in more detail the dimensions metamodeling layers (M) and concurrent engineering disciplines (C) which are part of our prototype implementation.

3.1. Metamodeling layers (M). Metamodeling is about models that describe other models. In our application scenario, a model is for example the hardware model of the automation devices used in our high-bay warehouse automation system shown in Fig. 2. This model includes specific devices used in this project like the Siemens Simatic CPU 317T-2 DP with its I/O modules. The metamodel of this hardware configuration model describes the common structure and the common attributes of hardware configuration models in general, i.e. the characteristics of all valid hardware configuration models. For example, the automation engineering model might include the mentioned item "Siemens Simatic CPU 317T-2 DP, Infeed" while the metamodel defines that the hardware configuration model includes elements of type CPU with attributes like vendor identification (Simatic CPU 317T-2 DP in our example) and symbolic name (Infeed in our example). This concept of metamodeling layers is adopted without modification from the Meta Object Facility (MOF) Specification [7] with its four layer metadata architecture. The four layers are called layer M0 up to layer M3, with layer M0 being the most concrete and M3 the most abstract layer (see Fig. 5).

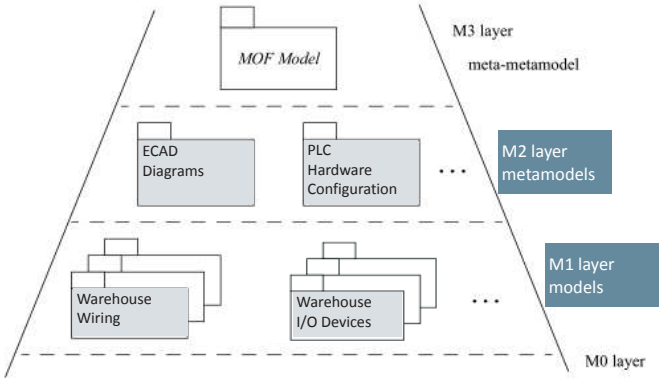

Fig. 5. Metadata architecture according to ISO/IEC 19502 [7]

In our example above, the "Siemens Simatic CPU 317T-2 DP, Infeed" is part of M1 layer, that describes the model of the automation system used for our specific high-bay warehouse system implementation (Warehouse I/O Devices on layer M1 in Fig. 5) while the M2 layer defines that the automation engineering tool (Simatic Step7 [5]) includes CPU modules to describe the hardware configuration (PLC Hardware Configuration on layer M2 in Fig. 5). In the next section we will use these two layers, M1 and M2, along the dimension M of the CMDAE hypercube together with the hypercube dimension C (Concurrent Engineering Disciplines) to describe the usage of the classification scheme. Layer M0 (the information layer) and layer M3 (the meta-metamodel layer) are discussed in detail in [7] but not considered in the sequel. In addition to the short introduction in this section, more information about metamodeling can be found in [8] and [9].

3.2. Concurrent engineering disciplines (C). The concurrent development of mechanical engineering, electrical engineering, and automation engineering requires data exchange about the devices used for machine automation, their characteristics, and their interaction with other machine modules. Within our application example, PLC programming is integrated with the concurrently running engineering activities of electrical engineering. Information of the PLC engineering project like hardware configuration, I/O addresses, and symbols is exchanged with the ECAD system. The integration of these concurrent engineering disciplines is described on the metamodeling layer (M2 layer according to Subsec. 3.1) in order to generate a tool integration environment for general use. This tool integration environment operates on the model layer (M1 layer according to Subsec. 3.1) to synchronize the engineering information (between ECAD and PLC engineering in our example, see Fig. 6) of the high-bay warehouse system engineering project.

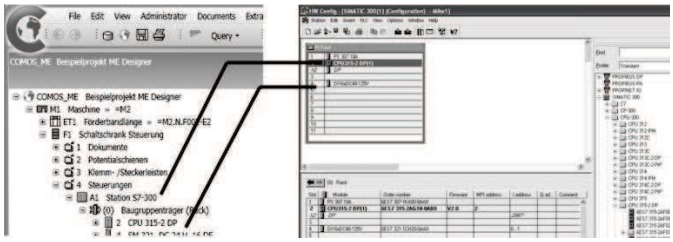

Fig. 6. Integration of electrical engineering and automation engineering as concurrent engineering disciplines

The metamodeling layer of the integration of concurrent disciplines (M2 layer) describes the relationships of elements in ECAD and PLC engineering. The electrical engineering of a machine like the high-bay warehouse automation system covers, for example, the control logic, power supplies, and safety devices. These functions are defined in the ECAD system by circuit diagrams with elements like terminals or PLC I/O modules [10]. Beside the logical interconnection of elements (defined by the wiring diagram), the location-oriented structure, shown in Fig. 6 at the left-hand side, reflects the physical placement of these elements e.g. in electrical cabinets or on specific mounting racks.

From a PLC engineering point of view, the location-oriented structure defines the relationship of the PLC program to the external system (the physical elements of the machine). In PLC engineering, the I/O modules and terminals are defined in the context of the hardware configuration of the PLC [11], shown in Fig. 6 at the right-hand side. Therefore, the integration between these concurrent disciplines must keep the information about the hardware configuration of automation engineering and the location-oriented structure of electrical engineering synchronized. For example, during the development of a new machine, changes of an I/O terminal

in electrical engineering must be propagated to automation engineering or changes of an I/O module must be propagated back to electrical engineering. This scenario is the operation of the integration environment on M1 layer, the engineering model of a specific machine (the high-bay warehouse system in our application scenario). The definition of the elements in each system, the I/O terminals and modules and their relationship is part of the metamodeling M2 layer. Beside change propagation, the definition of relationships between elements is also used for navigation and traceability purposes between the integrated systems.

4. Meta-modeling a tool integration

In Sec. 3, the new CMDAE hypercube classification scheme, which allows for the classification of complex integration scenarios, was introduced. Additionally, the separation of different dimensions was shown and where this scheme can be applied. In this section we will review the integration process for dimension C (concurrent) on layer M2 (metamodeling). We will discuss an approach, called *Tool-integration Environment* (TiE) that enables us to model complex data, to (semi-) automatically create source code for tool adapters that allow for processing specific tool data, and run bidirectional integrations on actual model instances.

Our goal is to integrate different modeling languages (M2 layer) and their instances, rooted in layer M1. This demand is derived from the requirement to edit different models simultaneously and exchange information in a bidirectional and incremental manner. A metamodel is used to represent the common scheme for all possible metamodel instances (i.e. models). Bidirectional data exchange between models can be achieved by different approaches. (Nearly) all concepts deal with integration by linking elements of at least two models. An example for such a model transformation approach is the specification *Query View Transformation* (QVT) [12] that is implemented by different techniques [12], e.g. by Triple Graph Grammars (TGGs) [13]. For a more detailed review of different model transformation approaches the reader is referred to [14].

During our research cooperation with Siemens AG, we decided to integrate two typical tools involved in automation engineering, electrical engineering with Comos ET [4] and PLC engineering with Simatic Step7 [5] (c.f. Sec. 2). Tool-integration enables engineers to exchange data between tools in a fast, reliable, and traceable manner. Furthermore, maintaining consistency is a major point for managing projects, where different models are developed concurrently. If information changes in one model, all other models have to be updated accordingly. Often it becomes hard to recognize conflicts or resulting points of change. Tool-integration also aims at helping engineers to cope with this difficult job. Thus, elements with mappable information in both tools have to be identified. These elements will become anchors in a tool-integration, where synchronization and transformation may start with. The first tool to be integrated is Step7 that is mainly used to program PLC components and setup a station logically. The second tool is Comos ET that describes the location-oriented structure of an (automation) project based on the standard IEC 61346 [15]. The workflow to build a tool-integration consists of the following steps at M2 layer (metamodeling):

1. Define a metamodel for the models manipulated by each tool (Comos ET & Step7).
2. Define an integration scheme between the tool's metamodels.
3. Generate platform specific integration code for tool specific adapters.

The workflow that uses the tool integration functionality consists of the following steps (M1 layer), described in Sec. 5:

1. Load the generated code into our Tool integration Environment (TiE).
2. Export the source model (e.g. the electrical engineering model of the high-bay area warehouse) from the engineering tool.
3. Run the integration, create the correspondence model with traceability linksand create the target model.
4. Import the target model (e.g. the automation engineering model of the high-bay area warehouse) to the engineering tool and manipulate the model data.
5. Synchronize models by recognizing the changes in the target model and update the source and correspondence model accordingly.

Building a (tool-) integration requires the development of at least three different adapters: One for each tool and a third one for the integration data itself. This general setup is depicted in Fig. 7 where these adapters provide interfaces to the integration framework. We shall review the adapters in detail as well as the integration framework.

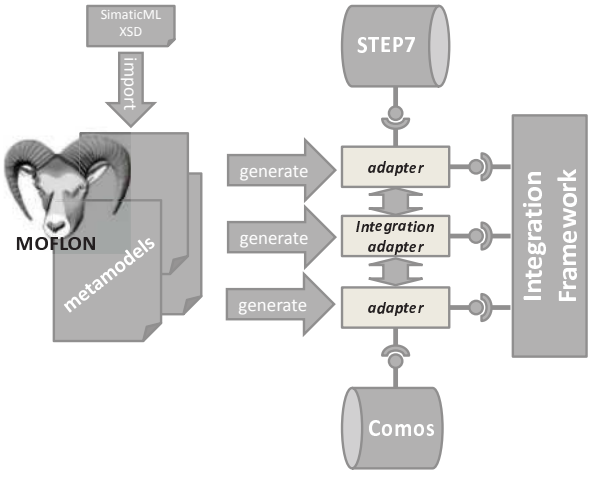

Fig. 7. Complete TiE architecture for tool-integration

First of all we have to model both tools (i.e. their data structure) in a MOF compliant way. Afterwards, a TGG specification is created, which defines a bidirectional mapping of both models. The metamodeling part is depicted on the left-hand side in Fig. 7. Further components of TiE are the gener-

ated adapters for accessing tools or data (center of Fig. 7) and the integration framework used for the application of integration rules, visualization, and model editing with a graphical user interface (right-hand side of Fig. 7).

The definition of a metamodel in general enables experts from different domains to define their specific languages reflecting their needs and a mapping between these languages. In tool-integration, the modeled languages are the appropriate representations of the data structure of the tools. Tool-integration does not define a new common language that must be adopted by the integrated tools. In this specific case we decided to model the tool information with MOFLON [16]. MOFLON is a Meta-CASE tool that features a MOF [17] compliant metamodel editor. CASE stands for *Computer Aided Software Engineering* and means that the user is supported with a graphical user interface and/or more convenient techniques to design its software. The prefix *meta* signals the ability to define conveniently metamodels with MOFLON. The created diagrams reflect the data structure a specific tool deals with. While defining the metamodel all participating domain experts describe their knowledge of their domain. Within this process and corresponding discussions, a common base is defined, where all experts find their knowledge reflected. As an example, supporting experts and experienced metamodel builders came up with an informal description of a *location-oriented model* from IEC 61346. As a result we defined the metamodel depicted in Fig. 8.

The metamodel in Fig. 8 shows how a project in a location-oriented model is set up and which components can be contained. A detailed instance of this model will be discussed in Sec. 5. Within our use case scenario, we have to build two adapters: one for the location-oriented structure models and one for Step7's hardware configuration (*HW-Config*). Both adapters are so-called *standard adapters* [18] since they are supposed to manipulate data that was processed and written by the tool itself. A standard adapter works on import and export data of a tool. The name *standard* is founded on experiences that have shown that the manipulation of this kind of data is the standard way to access a tool model. This means that a tool is used to edit the model instance after the tool adapter created or manipulated the instance itself. For example, our use case generates an XML document containing all the necessary project data that is further read and manipulated by Step7's XML import/export functionality for HW-Config data. This kind of adapters is mainly used when a tool API is not usable for various reasons. Therefore, we are not able to directly use the tool through its API to create instance of data objects and to manipulate them, but we have to bypass this drawback by accessing data that can be read from the tool. The implementations of our adapters provide a generic *Java Metadata Interface* (JMI [19]) to the outside world that is standardized with the help of Sun Inc. This interface enables us to retrieve and manipulate data within any model by using standardized methods. The other kind of adapters is called *online adapters*. These adapters provide the functionality to access a tool for creating and manipulating objects via a common interface. The actual adaption towards the tool's API has to be implemented manually. This type of adapters is used when a tool provides an API that allows for direct data manipulation through the tool itself. The adapter does not necessarily handle data persistence and manipulation. Therefore, all generic methods have to be adapted to use the API. In this case, generic stands for methods which provide the same interface and behavior independent of the model instance they are working on. Since this implementation is tool dependant, parts of the source code have to be implemented manually.

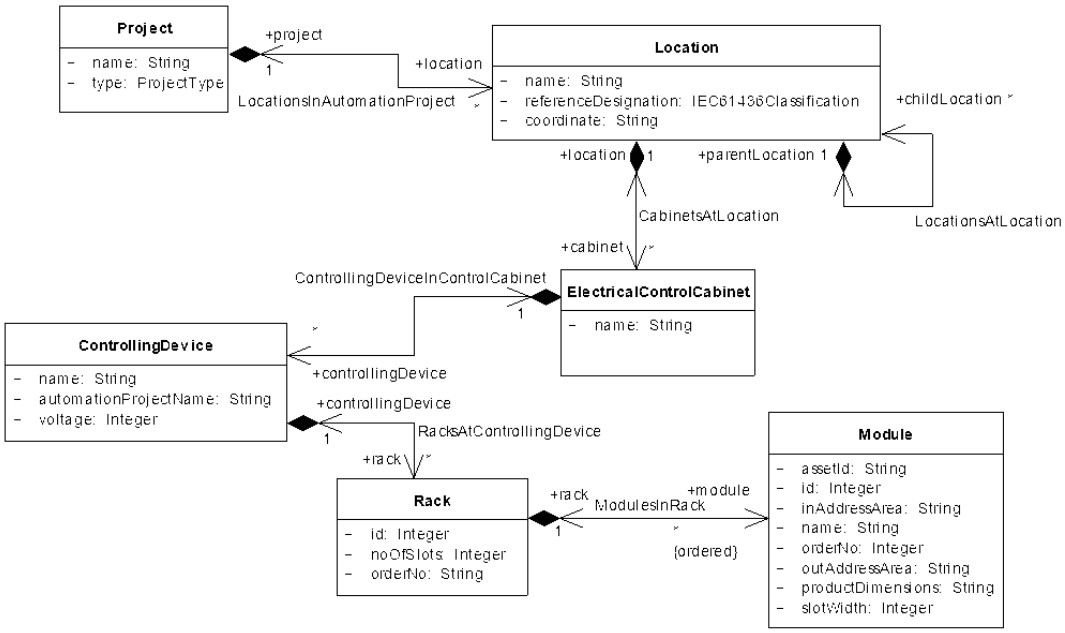

Fig. 8. Location-oriented model metamodel in correspondence to IEC 61346 (Ref. 15)

The second tool to integrate is Step7. In this integration scenario, we decided to model the subpart HW-Config of Step7 that deals with the hardware setup of an automation project. Siemens provides for its Step7 distribution an XML schema definition (XSD) for the export function of HW-Config data. This schema is called SimaticML. Since XSDs describe a language on the same layer as a MOF compliant metamodel does, we were able to transfer the concepts from the *SimaticML* XSD, depicted in Fig. 9, directly into MOFLON. By providing MOFLON with an XSD import function we are able to import an XSD file into MOFLON and to translate this into the appropriate MOF metamodel representation. Furthermore, the derived adapter from this scheme will be able to read and write natively all XML files defined by the XSD.

Figure 9 depicts the MOF compliant metamodel corresponding to the XSD schema of SimaticML documents. A ProjectT object contains a ProjectObjectListT object that represents a list that further contains all DeviceT objects. A complete instance of this model will be presented in Sec. 5. The next step is to describe the actual integration based on both metamodels introduced so far. This process involves the experts representing the participating domains. The integration in this case is done again with MOFLON that also provides the functionality to specify model integrations with TGGs. This is a common technique for modeling integration as explained in [20] and [21]. A TGG visually expresses the relationship between model elements of two domains with a so-called *link type*. Figure 10 (top) depicts the relationship between a Project object and a ProjectT object via a ProjectLink object.

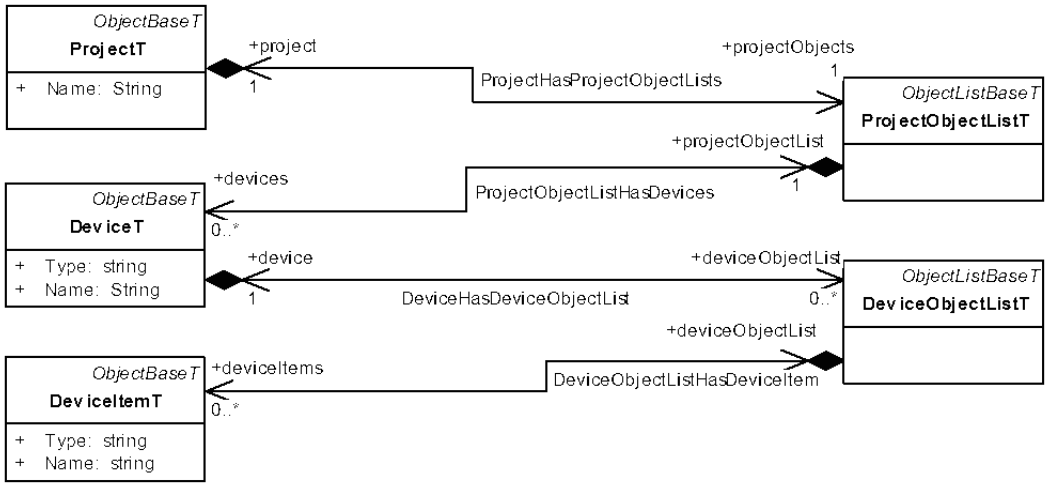

Fig. 9. Excerpt from SimaticML metamodel

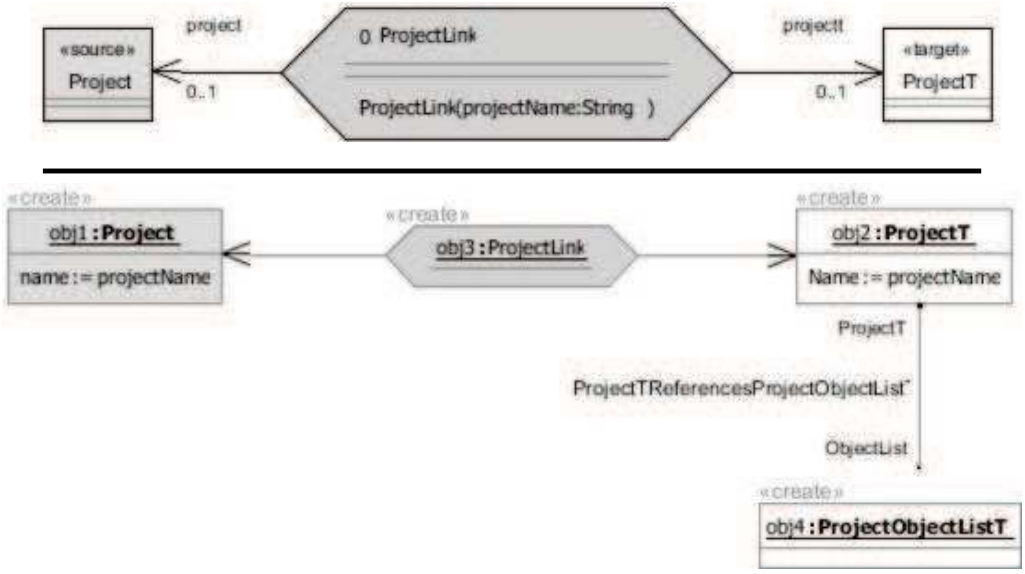

Fig. 10. ProjectLink definition and appropriate TGG rule

On the bottom of Fig. 10, a TGG rule is shown that relates all elements on the left to elements on the right. Further information like element names are set synchronous and an additional link is created for traceability. A TGG rule expresses in a declarative way, which elements will be set in correspondence and what prerequisites are needed to establish such a correspondence. For example, a ProjectT object in Step7's HW-Config must exist before a DeviceItemT object, e.g. a CPU, may be added. Additional information like parameter values for new objects is also set in a specific rule. A TGG consists of rule expressions used for deriving multiple so-called *translators* that explicitly provide integration features such as forward translation or consistency checking (for more information on translators refer to [13]). A TGG project refers to both to-be-integrated metamodels and additionally contains integration specific elements, e.g. a link type reflecting the generic relationship between two elements. Each link type is also provided with different integration methods (e.g. forward translation, consistency checking etc.). Domain experts supported by software engineers identify related elements in each tool model and express the relation with TGG links.

After both tool metamodels and the integration have been specified, the concrete adapter implementation can be derived (semi-) automatically and has to be manually enriched with additional implementation details. The functionality of this implementation will be explained in detail in Sec. 5 since it represents a part of the model integration functionality used on layer M1. Altogether, several components are required to build a tool-integration for a pair of metamodeled languages. The approach we use is built on top of different tools and techniques that shape the complete Tool-integration Environment (TiE). Figure 7 depicts this environment used as a prototype implementation within this research cooperation.

As already mentioned, our work is MOFLON-centered. In order to reduce manual efforts, we use some generic interface technology that allows us to access all tool data with the same methods. In Fig. 7 different adapters are connected to the integration framework. Instead of implementing different data access methods in the integration framework for each kind of adapter, every adapter data can be accessed with the same basic methods. Further advantages of using such generic interfaces is the possibility to provide different services for all kind of models that support such a generic model interface – depicted in Fig. 11.

We implemented a generic persistence service that allows us to read and write model instances into an *XML Metadata Interchange* (XMI) [22] file. Additionally, we implemented a generic difference detection service (Diff) that finds changed data in a model automatically and is able to provide this information to the three-way merge service that may build upon the differences a new model version. Further services like database retrieval are in a design phase.

The introduced tool-integration environment (TiE) is a prototype implementation that allows us to model tools' data structures and execute integration rules. Although, the integration environment is a prototype, we believe that the architecture of our TiE can be reflected in a commercial tool for productive use. Furthermore, we see TiE as a playground for developing new ideas and concepts and testing them with a concrete implementation. Thus, TiE enables us to implement scenario prototypes, as described in this contribution, and evaluate new ideas and concepts. Future commercial model integration solutions may benefit from this expertise and select different aspects from TiE.

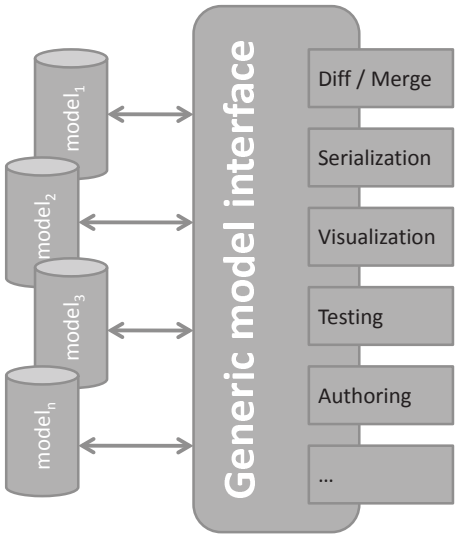

Fig. 11. Model interface services

5. Proof-of-concept prototype

In Sec. 4 we had a look at our prototype scenario from the perspective of layer M2. For this section we will have a close look at the functionality of our generated tool adapters and the way the integration is applied according to Subsec. 2.1. This more concrete layer can be referred to as M1, according to the classification scheme described in Sec. 3.

We created all necessary components to apply our tool-integration so far. The use case we apply next is defined as follows and depicted in Fig. 12: First of all, we start with a single location-oriented model instance that contains a single project with some sub-objects (Fig. 12 (1)). The next step is to this data into a HW-Config data instance and associate related elements with traceability links (2). After importing our HW-Config data in Step7 we add and manipulate information and use the changed data (3) as input for a difference recognition mechanism (*diff*) (4). The result of the diff process is a collection of changes applied to the HW-Config data. Finally, changes are taken into account when data backwards (5). The result is a modified location-oriented model instance (6).

We start with the frontend application of our integration framework, shown in Fig. 13. This application is called *Integrator* and allows loading JMI compliant adapters. "Loading" means that we register a metamodel, represented by the JMI adapter, and that we are able to create and edit any instance of this metamodel. For integration purposes, we need to load both tool adapters and the integration adapter.

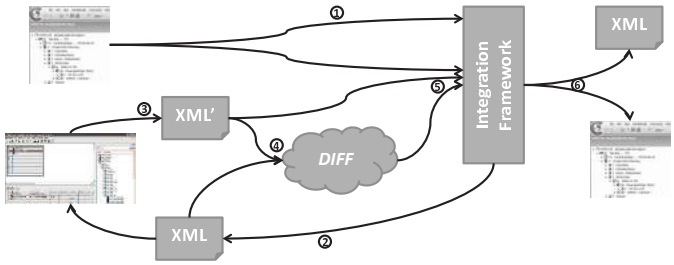

Fig. 12. Workflow sketch for the running example

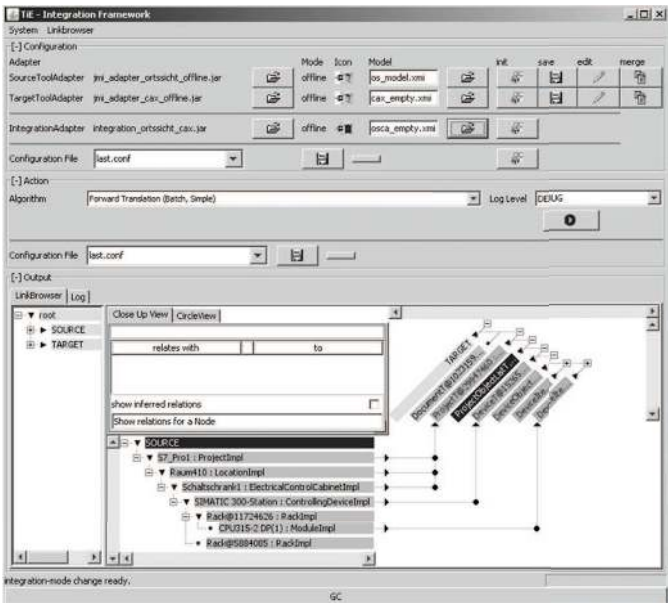

Fig. 13. Integrator with traceability links after applying a forward

The integration scenario starts with the creation of a location-oriented model instance (c.f. Fig. 12 (1)) in the tool Comos (screen dump in Fig. 12 top-left). This instance contains one Project object with two Location objects. The first Location object contains a ControllingDevice object with a Rack object and a CPU in the sense of a Module object. The second Location object is empty. Figure 14 depicts the object diagram of our running example after applying all translation steps from Fig. 12.

On the left-hand side of Fig. 14 the location-oriented model instance is depicted. A Project object contains two Location objects, from which one Location object contains additional components. On the right-hand side the HW-Config model instance is depicted that is composed of a ProjectT object and its ProjectObjectListT object with an additional DeviceT object and so on. Between both model instances the correspondence graph is depicted that connects related elements of both models. These links are the traceability links for change recognition and consistency checking. By invoking the forward translation all rules are executed that find a match in the location-oriented model instance and create the corresponding elements in the HW-Config data. Additionally, correspondence links between these elements are created that preserve traceability. Figure 13 depicts in the lower part the matrix representation of a similar object diagram compared to Fig. 14. While executing the forward translation an internal algorithm picks element by element from the input model that has to be translated. With each element type a number of appropriate rules is connected from which a certain rule is taken and executed. This process is repeated for all elements in an appropriate order until they are completely translated or an error occurs. For a detailed description of the rule application algorithm and correlated concerns the reader is referred to [23].

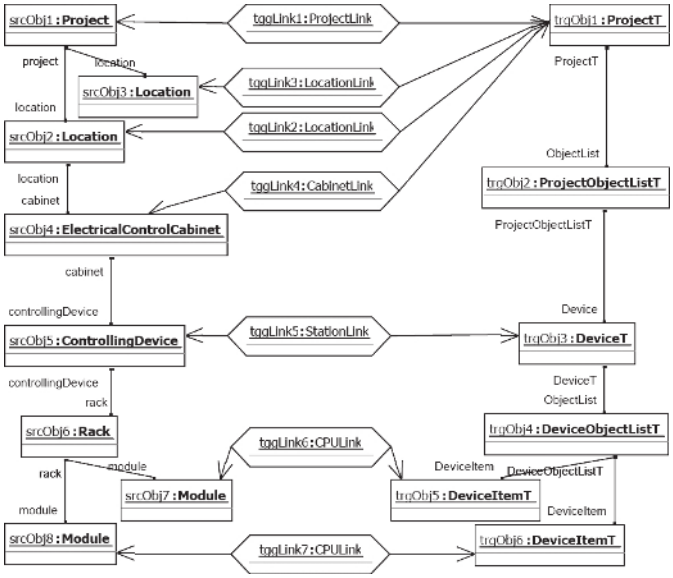

Fig. 14. Object diagram of an integrated location-oriented model instance and an HW-Config model instance

The next step in our integration scenario is to import the just created HW-Config data as a XML representation into Step7 (c.f. Fig. 12 (2), screen dump lower left). To this end, Step7 provides an import function that allows previewing the result by highlighting changed or just created content. The user is allowed to choose which information has to be imported. The data is converted into an internal representation during import. This leads to information loss when additional information, not relevant for Step7's HW-Config, is held by certain objects. In Step7, changes are applied according to our use case: An additional Input/Output module is created and the name of the CPU is changed.

The changed data is saved to an XML file (c.f. Fig. 12 (3)) by using the export functionality of Step7. Now, two versions of HW-Config data are in place: The initially integrated version and the changed document from Step7. The diff, as part of our generic services depicted in Fig. 12, is used for detecting changes (c.f. Fig. 12 (4)) between the initial HW-Config data and the new version and saves them to a list of all changed elements. The way these changes are computed is not important here and the algorithms used in TiE apply incremental update methods driven by this changelist. This allows us to think of other ways to recognize changes, such like throwing events or manual interaction. In consequence, the backward translation must only consider modified elements in order to achieve consistency between both models (c.f. Fig. 12 (5)). Anyway, computing such a change list may be difficult, when

tools do not support unique permanent identifiers for objects. Step7's export mechanism does not provide an ID for any of its objects. From this a standard difference recognition method may treat all objects as new objects. In our specific use case, we merge the new information by hand into the old XML file and therefore provide our diff with correct information only.

The input for our backward translation is the initial location-oriented model instance and the modified HW-Config XML file (Fig. 12 (5)). The result of the backward translation is the creation of a new Module object in the location-oriented model instance and changing the name of the CPU accordingly. The data representation is incrementally updated with the new information. The creation of the Module object is initiated since there is no counterpart in the already related model. On the other hand the propagation of the changed CPU name does not result in a new object though there is also no counterpart with the same name but a previously established traceability link that enables the algorithm to recognize changes in an existing object. Additionally, a new traceability link between the new Module and the DeviceItemT object is established. Both model instances are now consistent; that means they are containing the equivalent information and can be further used and evolved in their own tools (c.f. Fig. 12 (6)).

6. Related work

Integrated development environments were a research topic for the past decades. Different approaches have been created [24–27]. According to our CMDAE classification scheme (c.f. Sec. 3) and the defined tool-integration requirements (c.f. Subsec. 2.1) we are now able to classify such approaches and tools. Within this section we restrict our focus to approaches and tools that were important during our research cooperation in terms of experiences, concepts, and technology. Therefore, we do not claim to provide a complete overview of all approaches and tools available. Table 1 arranges the approaches and tools according to the dimension they deal with.

Existing standards by the OMG relevant for MDA do not give a complete overview from the megamodeling, described in [28] perspective. Only the MOF Facility and Object Lifecycle (MOFFOL) specification defines an overall metamodel for the environment of any metamodel and instances. But MOFFOL contains no classification scheme and is restricted to models based on MOF. QVT, OMG's standard of a model transformation language, deals implicitly with dimension C and A. The MOF Versioning and Development Lifecycle is an addition to MOFFOL to deal with different evolution steps of models. Several other publications deal with a single dimension of the hypercube. In [29] versioning in a model environment is presented as a service similar to the concurrent version system (CVS) [30]. As a result, our hypercube is a classification scheme for models in the broadest sense. In [31] translations are classified independent of the models they are applied to. Our approach for the classification of models will help users in selecting tailored translations based on the user's choice of involved models. The so-called megamodeling is close to our concepts of an infrastructure supporting the CMDAE hypercube. Our approach also supports some organizational criteria that might be useful for different services like visualization or browsing [32]. The environment supporting the CMDAE hypercube provides tailored services. For example, the way of translating changes of a model might depend on its classification. Tools like Reqtify [33] are able to deal with different domains since they support domain specific tailoring in means of tool specific adaptions. Altova Mapforce [34] and XLinkit [27] both support metamodeling implicitly. This means meta-information, provided via XSD files, can be used for the describing consistency rules or the data mapping. Thus, on instances of such XSD files, i.e. corresponding XML files, data conversion (Mapforce) or rule-based consistency checks (XLinkit) can be applied.

Toolnet [24] had a focus on read-only access on related tools. Additionally, all required software components were written manually. Our approach, as a successor of Toolnet, provides read/write access to related tools as well as accessing information directly from storages like databases or files. Metamodeling enables semi-automatic creation of required software components. Especially the integration of engineering disciplines within machine and plant engineering has been investigated in several projects. Planning of manufacturing cells is often based on a common database as described in [35]. The integration of the data models of different engineering tools in such a database is assumed as a manual development process and not based on CASE methods as in CMDAE. Approaches for synchronizing models based on TGGs are summarized in [23]. One approach concerning synchronization of engineering models of manufacturing control systems is introduced by [36]. Reconciliation between plant engineering and plant simulation based on TGG is implemented by [37]. Both TGG applications are in different application domains and do not use common standards like [7] and [19].

Table 1
Classification of approaches and tools according to the CMDAE hypercube (c.f. Sec. 3)

Hypercube Dimension	Approach or Tool								
	Reqtify	Mapforce	Biztalk	FÖDERAL	XLinkit	TiE	QVT	MOFFOL	MOFVDL
Dimension C	X	X	X		X	X	X		
Dimension M		(X)	X	X	(X)	X		X	
Dimension D	X								
Dimension A		X		X			X		
Dimension E	X		X						X

Table 2
Classification of approaches and tools according to integration requirements (c.f. Subsec. 2.1)

Integration Requirement	Approach or Tool					
	Reqtify	Mapforce	Biztalk	FÖDERAL	XLinkit	TiE
Translation		X	X			X
Automatic Translation		X	X	X		X
Incremental Translation			X			(X)
Automatic Incremental Translation			X			(X)
Consistency Checking	X	X		X	X	X
Late repair (Engineer decides)	(X)		(X)		(X)	
Backtracking (Play through different variants)						
(Explicit) Traceability Links	X	X				X
Customization of Traceability Links (own datatypes)	X					

Table 2 classifies tools and approaches according to the integration requirements (from Subsec. 2.1) they fulfill.

Several industrial tools aim for supporting the integration of different information sources in engineering processes. MapForce is part of a tool set from Altova. It provides visual specification of unidirectional data mappings based on two XSDs. The mapping is stored as an XML file itself. MapForce supports code generation based on the mapping for various technologies and languages like C, Java or XSLT. In comparison, our approach provides visual specification of bidirectional mappings. As already mentioned, we are able to (semi-) automatically translate between a MOF metamodel and an XSD representation. This enables us to use metamodels created within our approach also in MapForce. The relation between TGGs and MapForce mapping is part of our further investigations. MapForce and our approach are both not domain specific. Reqtify is a tool from Geensys, which provides integration of many different tools. Reqtify has its main focus on requirement engineering and traceability. Reqtify, therefore, supports so-called interfaces to support several other tools as information sources. Another example for a tool integrating several other tools by a wide range of connectors is Microsoft Biztalk [38]. As Reqtify does, Biztalk provides those connectors as part of the product. In contrast to Biztalk and Reqtify our approach contains a workflow for creating tool interfaces in a semi-automatic and generic way.

Mechatronic engineering according to FÖDERAL also [39] does not require the integration of all data in a common database but implements a separate database that links information between the related engineering tools. In contrast to TiE the adapters and translations from existing engineering tools to the common database of FÖDERAL have to be created manually. Another direction is not the integration of the data models but the integration of the related engineering disciplines in a common tool as proposed by [40]. This approach is feasible for the integration of some engineering disciplines but not in general for an existing tool environment. Additionally, XLinkit can be extended with a methodology to repair model instances. Starting from the point that inconsistencies do no harm in general, the engineer is responsible for choosing the right actions.

7. Summary and open challenges

In this contribution, we showed that the tool-integration approach with MOF compliant metamodeling and TGG as an integration technique is an appropriate means for the integration of automation engineering tools. We developed a prototype that reflects the typical steps for tool-integration. In parallel, a classification scheme was developed that can be used to analyze complex integration scenarios. The integration between concurrent disciplines was investigated in the exemplary field of integrating ECAD and PLC. We defined MOF compliant metamodels for both domains and additionally integrated them with a TGG. The use of MOFLON as an editor and code generator and TGG as an integration technique proved to be a highly potential combination for integration tasks.

Nevertheless, problems were also encountered when integrating tools in the complex domain of automation engineering. Currently, it takes a lot of fundamental knowledge about metamodeling for a non-software engineer to model complex data structures. The metamodeling process is hard to cope with for people not familiar with software engineering principles. Domain engineering experts are needed that provide support to describe the tool models. Discussions showed that one can think of metamodeling as a service-business concept, where domain-specific engineers and metamodel experts sit together and establish a common metamodel for certain tools or data. Furthermore, the use of library and template concepts may support engineers to re-use modules and concepts on different levels. Building the metamodel is only the starting point for a tool-integration. Since integration requirements may change during project phases it is necessary to enable automation engineers to modify the integration on their own. This process is hardly intuitive, but ideas how to ease integration modeling already exist. The future goal is to facilitate the build process of (tool-) integration for all participated engineers on different levels. Recently, we explored ideas and concepts for dimensions C and M (c.f. Fig. 4). The next dimension to be investigated is abstraction (A). Here different concepts already exist that deal with information hiding and detailed information. ViewTGGs (VTGGS) are a theoretical concept to cope with abstraction based on our well-known TGG approach [41] that may enable us to abstract between

different metamodeling layers (c.f. Subsec. 3.1). Abstraction on the same layer in the sense of platform specific information vs. platform independent information can be achieved with stratification [42]. Furthermore, other dimensions like evolution (E) have to be our focus when it comes to integration of models into product lifecycle management tools (PLMs). Concrete next steps for improving our approach will be:

- Build a plugin for a commercial tool, e.g. Enterprise Architect, that allows you to create MOF and TGG compliant diagrams and export them to a MOFLON API that takes care of code generation.
- Decouple code generation from platform specific aspects.
- Improve usability of the integration execution algorithms.

We are also planning to improve MOFLON and TGG theory with additional concepts according to these user requirements. Abstraction and modularization of modeling components will be a large area of further research. New concepts have to be established that enable users to hide and abstract from details. The modularization of components will enable engineers to model typical problems in a one-time-effort and re-use them. We are researching techniques that also parameterize re-usable components in order to provide additional domain-specific information.

Together with Siemens AG, we want to drive the CMDAE approach to a common base of meta-modeling complex systems in automation engineering and therefore benefit from efficiency and quality enhancement known from model-based software engineering.

Acknowledgements. The work of Marius Lauder is supported by the 'Excellence Initiative' of the German Federal and State Governments and the Graduate School of Computational Engineering at Technische Universität Darmstadt.

REFERENCES

[1] VDI-Gesellschaft Entwicklung Konstruktion Vertrieb, *Entwicklungsmethodik fuer mechatronische Systeme*, VDI 2206, 2004.

[2] VDI-Gesellschaft Fördertechnik Materialfluss Logistik, *Digitale Fabrik – Grundlagen/Digital factory – Fundamentals*, VDI 4499, 2007.

[3] International Electrotechnical Commission (IEC), *Programmable Controllers – Part 3: Programming languages*, IEC 61131–3, 2003.

[4] Comos Industry Solutions GmbH, *Comos ET*, http://www.comos.com/elektrotechnik.html?&L=1 (2009).

[5] Siemens AG, *SIMATIC STEP7 Programming Software*, http://www.automation.siemens.com/simatic/industriesoftware/html_76/products/step7.htm (2007).

[6] IEEE Computer Society, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, IEEE 1471, 2000.

[7] ISO/IEC JTC 1/SC 32 Daten-Management und Daten-Austausch, *Information technology – Meta Object Facility (MOF) Specification*, ISO/IEC 19502, 2005.

[8] Object Management Group, *OMG Model Driven Architecture*, http://www.omg.org/mda/ (2009).

[9] I. Weisemöller and A. Schürr, "A comparison of standard compliant ways to define domain specific languages", *Models in Software Engineering. Workshops and Symposia at MODELS 2007, Reports and Revised Selected Papers* 1, 47–58 (2008).

[10] IEC/TC 3 Information structures, documentation and graphical symbols, *Preparation of Documents Used in Electrotechnology – Part 1: Rules*, IEC 61082–1, 2006.

[11] *Configuring Hardware and Communication Connections with STEP7. Manual*, 6ES7810-4CA08-8BW0 (2006).

[12] Object Management Group, *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification OMG Available Specification*, http://www.omg.org/spec/QVT/1.0/PDF/ (2009).

[13] A. Schürr, "Specification of graph translators with triple graph grammars", *Graph-Theoretic Concepts in Computer Science. 20th Int. Workshop, WG '94 Herrsching, Germany, Proceedings* 1, 151–163 (1995).

[14] T. Mens, K. Czarnecki, and P. Van Gorp, "A taxonomy of model transformations", *Dagstuhl Seminar Proceedings* 1, CD-ROM (2005).

[15] IEC/SC 3B Dokumentation, *Industrial Systems, Installations and Equipment And Industrial Products – Structuring Principles And Reference Designations – Part 1: Basic Rules*, IEC 61346–1, 1996.

[16] C. Amelunxen, F. Klar, A. Königs, T. Rötschke, and A. Schürr, "Metamodel-based tool integration with MOFLON", *30th Int. Conf. on Software Engineering* 1, 807–810 (2008).

[17] Object Management Group, *Meta Object Facility (MOF) 2.0 Core Specification* OMG Available Specification, http://www.omg.org/cgi-bin/doc?formal/06-01-01.pdf (2006).

[18] F. Klar, S. Rose, and A. Schürr, "TiE – a tool integration environment", *Proc. 5th ECMDA Traceability Workshop* 1, 39–48 (2009).

[19] *Java Metadata Interface (JMI) Specification. JSR 040 Java Community Process*, 2002.

[20] E. Kindler and R. Wagner, *Triple Graph Grammars: Concepts, Extensions, Implementations, and Application Scenarios*, University of Paderborn Paderborn, 2007.

[21] A. Königs and A. Schürr, "Tool integration with triple graph grammars – a survey", *Proc. SegraVis School on Foundations of Visual Modelling Techniques* 1, 113–150 (2006).

[22] ISO/IEC JTC 1/SC 32 Daten-Management und Daten-Austausch, *Information Technology – XML Metadata Interchange (XMI)*, ISO/IEC 19503, 2005.

[23] A. Schürr and F. Klar, "15 years of triple graph grammars – research challenges, new contributions, open problems", *4th Int. Conf. on Graph Transformation* 1, 411–425 (2008).

[24] F. Altheide, S. Dörfel, H. Dörr, and J. Kanzleiter, "An architecture for a sustainable tool integration", *Workshop on Tool Integration in System Development, TIS 2003* 1, 29–32 (2003).

[25] M. Nagl, *Building Tightly Integrated Software Development Environments: The IPSEN Approach to Building Tightly Integrated Software Development Environments. The IPSEN Approach*, Springer, Berlin, 1996.

[26] A. Schürr and H. Dörr, "Special section on model-based tool integration", *Software and Systems Modeling* 4 (2), 109–170 (2005).

[27] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein, "xlinkit: a consistency checking and smart link generation service", *ACM Transactions on Internet Technology* 2 (2), 151–185 (2002).

[28] J. Bezivin, F. Jouault, and P. Valduriez, "On the need for megamodels", *Proc. Workshop on Best Practices for Model-Driven*

Software Development 19th Annual ACM Conf. on Object-Oriented Programming, Systems, Languages, and Applications 1, CD-ROM(2004).

[29] G. Kappel, G. Kramler, E. Kapsammer, T. Reiter, W. Retschitzegger, and W. Schwinger, "ModelCVS – a semantic infrastructure for model-based tool integration", *Technical Report* 1, http://www.modelcvs.org/papers/0705.pdf.(2007).

[30] G. Kramler, G. Kappel, T. Reiter, E. Kapsammer, W. Retschitzegger, and W. Schwinger, "Towards a semantic infrastructure supporting model-based tool integration", *GaMMa '06: Proc. 2006 Int. Workshop on Global Integrated Model Management* 1, 43–46 (2006).

[31] K. Czarnecki and S. Helsen, "Classification of model transformation approaches", *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture* 1, CD-ROM (2003).

[32] A.W. Brown, D.J. Carney, E.J. Morris, D.B. Smith, and P.F. Zarrella, *Principles of CASE Tool Integration*, Oxford University Press, Oxford, 1994.

[33] Geensys, *REQTIFY*, *Effective Solution for Managing Requirements Traceability and Impact Analysis across Hardware and Software Projects Lifecycle*, http://www.geensys.com/?Outils/Reqtify (2009).

[34] Altova, *MapForce*, http://www.altova.com/products/mapforce/data_mapping.html (2009).

[35] J. Kiefer, „Mechatronikorientierte Planung automatisierter Fertigungszellen im Bereich Karosserierohbau", *Phd Dissertation*, Univ. des Saarlandes Lehrstuhl für Fertigungstechnik, Saarbrücken, 2007.

[36] R. Wagner, „Inkrementelle Modellsynchronisation", *Phd Dissertation*, University of Paderborn, Paderborn, 2009.

[37] S.M. Becker, S. Herold, S. Lohmann, and B. Westfechtel, "A graph-based algorithm for consistency maintenance in incremental and interactive integration tools", *Software and Systems Modeling* 3, 287–315 (2007).

[38] Microsoft, *Microsoft BizTalk Server*, http://www.microsoft.com/biztalk/en/us/default.aspx (2009).

[39] *Baukastenbasiertes Engineering mit Föderal. Ein Leitfaden für Maschinen- und Anlagenbauer*, VDMA Verl., Frankfurt am Main, 2004.

[40] F.M. Grätz, "Teilautomatische Generierung von Stromlauf- und Fluidplänen für mechatronische Systeme", *Phd Dissertation*, Utz Herbert, München, 2006.

[41] J. Jakob, A. Königs, and A. Schürr, "Non-materialized model view specification with triple graph grammars", *Proc. Int. Conf. on Graph Transformations*, *Lecture Notes in Computer Science (LNCS)* 4178, 321–335 (2006).

[42] C. Atkinson and T. Kühne, "Aspect-oriented development with stratified frameworks", *IEEE Software* 20 (1), 81–89 (2003).