

# Automated Derivation of Configurations for the Integration of Software(+) Engineering Environments

Stefan Biffel, Richard Mordinyi and Thomas Moser

Christian Doppler Laboratory “Software Engineering Integration for Flexible Automation Systems”

Vienna University of Technology

Vienna, Austria

{stefan.biffel, richard.mordinyi, thomas.moser}@tuwien.ac.at

**Abstract**—Today’s systems integration technologies enable the integration of (software+) engineering environments to support engineering processes across domain and tool boundaries. These engineering processes heavily rely on manual configuration of integration frameworks, resulting in costly, time-consuming, and error-prone human work. In this paper, we introduce an extended model-driven approach for the automated derivation of integration technology configurations for supporting engineering processes. This allows both an efficient and effective derivation of initial configurations, as well as easy adaptations of existing configurations in case of changed engineering processes. Based on a standard software engineering process, we show the feasibility of the proposed approach and discuss the advantages and limitations for software(+) engineering.

**Keywords:** *engineering domains; systems integration; model-driven approach; automated configuration*

## I. INTRODUCTION

Typical large-scale engineering projects, like power plants or car manufactures, involve the cooperation of a wide range of engineering systems and tools that use different technical platforms and heterogeneous data models (e.g., mechanical, electrical, and software engineering [5]). This kind of cooperation is called “(software+) engineering projects”, since software engineering provides additional values to software-intensive systems and also depends on the seamless collaboration with other engineering fields. Today’s system integration technologies are suitable to bridge most of the technical and semantic gaps between these (software+) engineering systems and tools. However, error-prone and time-consuming human work (e.g., manually copying information from one to another tool) is needed to handle integration concerns at the interfaces of different engineering disciplines. An example for reducing the needed human work, and thus for the integration of heterogeneous engineering disciplines, is the Engineering Service Bus (EngSB) [6], which is a process-oriented framework for both technical and semantic integration of heterogeneous software tools.

The configuration of such system integration technologies is time-consuming and a complex manual task that can only be done by designated integration experts (IE) [12]. Complexity arises from the integration of a high number of distributed and heterogeneous engineering tools (e.g., different accessibility

methods for each tool), different tool data formats (e.g., open standards vs. proprietary data formats), and the need for on/offline capabilities (e.g., high availability of server-based tools vs. limited availability of end-user tools). The manual configuration of the used integration technology with respect to these complexity issues is often inefficient and incorrect. In order to reduce manual configuration efforts and defects, as well as to increase the efficiency of integration, automated derivation of configurations can be used. There exist methods for the automated derivation of configuration based on the Model-Driven Architecture (MDA) [15] paradigm, such as the Model-Driven System Configuration (MDSC) approach [16]. The MDSC approach explicitly models a) the semantics of integrated engineering tools’ requirements and capabilities [17]; and b) the connectors and data transformations between heterogeneous engineering tools [16], to simplify systems integration. Based on these semantic models, the MDSC approach is capable of automatically deriving integration technology configurations.

In this paper, we describe an extended version of the MDSC approach applied to the integration of heterogeneous tools originating from (software+) engineering disciplines. In contrast to the traditional MDSC approach, the extended MDSC (eMDSC) approach uses semantically modeled requirements of pre-defined engineering processes to a) select a set of suitable engineering tools providing the required capabilities; and b) to derivate configurations for the integration technology accordingly. The eMDSC approach consists of two major steps: in the first step, requirements of engineering processes are matched against capabilities of so-called tool domains [6], which abstract tools providing similar functionality. The outcome of this matching is an ordered set of tool domains required for the execution of a specific engineering process. In the second step, for each of these required tool domains a so-called tool instance, i.e., the actual engineering tool, is selected. Again, the engineering process requirements regarding a specific engineering domain are matched against the capabilities of a certain tool instance in order to identify suitable engineering tools for an engineering process step.

We evaluate the proposed eMDSC approach using a well-known software engineering process, the Continuous Integration & Test (CI&T) process [9]. The results of the

evaluation show that the proposed approach is efficient and effective, regarding both the effort needed for setting up the integration environment as well as the number of error sources. Furthermore, tool domains enable easy adaptations of existing integration solutions by allowing the efficient exchange of similar tools without affecting the existing engineering process.

The remainder of this paper is structured as the following: Section II reports related work regarding common technical and semantic integration approaches as well as the MDSC. Section III presents research issues concerning the integration in (software+) engineering domains, while section IV describes an industrial use case. Section V explains the eMDSC approach applied for (software+) engineering domains. Section VI presents evaluation results and section VII discusses them. Finally, section VIII concludes the paper and presents future work.

## II. RELATED WORK

This section summarizes related work on system integration technologies for technical and semantic integration as well as for software and systems integration in automation systems engineering, and on model-driven system configuration.

### A. System Integration Technologies

Current developers of software systems use a wide range of tools from software vendors, open source communities, and in-house developers. Getting these tools to work together to support a development process in an engineering environment remains challenging as there is a wide variety of standards these tools follow [13]. Any integration approach has to address the levels of *technical heterogeneity*, i.e., how to connect systems that use different platforms, protocols, etc., so they can exchange messages [7, 12]; and *semantic heterogeneity*, i.e., how to translate the content of the messages between systems that use different local terminologies for common concepts in their domain of discourse, so these systems can understand each other and conduct a meaningful conversation [1, 11, 17].

**Basics of technical integration.** Technical integration follows message-based patterns [12] to connect a series of technically heterogeneous and distributed systems. The communication between these systems is in many cases event-based and sometimes request/response-based. Message-oriented middleware (MoM) and an “Enterprise Service Bus” (ESB) [7] provide the infrastructure for physically and logically connecting technically heterogeneous systems with technical integration features such as message processing (like routing, filtering and enriching messages) and a service registry (a directory of currently available services, their names, interface and behavior descriptions, and location to bind and invoke) [2], and thus are the foundation for engineering process services on domain level. To efficiently embed these infrastructure tools an engineering project with resource-constrained, mobile, and low-cost environments, there are several lightweight open source ESB and related middleware products available [21].

**Basics of semantic integration.** Semantic integration is defined as the solving of problems resulting from the intent to

share data across disparate and semantically heterogeneous data [10]. These problems include the matching of ontologies or schemas, the detection of duplicate entries, the reconciliation of inconsistencies, and the modeling of complex relations in different data sources [20]. One of the most important and most actively studied problems in semantic integration is establishing semantic correspondences (also called mappings) between vocabularies of different data sources [8]. The application of ontologies as semantic web technologies for managing knowledge in specific domains is desirable. Moser *et al.* [17] introduced the Engineering Knowledge Base (EKB) framework as a semantic web technology approach for addressing challenges originating from data heterogeneity that can be applied for a range of domains, e.g., in the production automation domain [17] and also Software Engineering.

**Software and Systems integration in Automation Systems Engineering.** Integration of engineering systems is a challenge as (particularly in the automation industry) typically a broad range of engineering tools from different vendors are used to solve specific problems [22]. Tools within one vendor are sometimes integrated to exchange data, but hardly between vendors. APIs and exchange formats often do not follow established (open) standards. Therefore the *AutomationML*<sup>1</sup> project provides a standardized XML data exchange basis for data integration between multi-vendor automation systems engineering tools as foundation systematic information exchange between engineering models. The *Medeia*<sup>2</sup> project develops an automation component model concept as foundation for knowledge exchange between semantically heterogeneous domain-specific engineering models [14]. The results of these projects become essential for engineering teams that have a technically well-integrated environment but need to reconcile different semantic approaches in the engineering team.

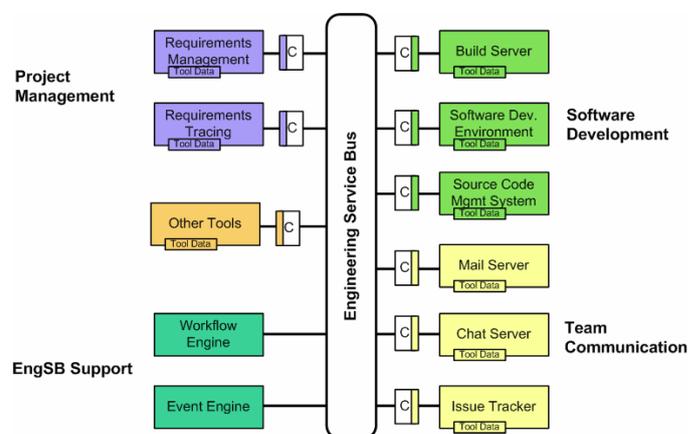


Figure 1. Overview EngSB platform [6].

**Engineering Service Bus (EngSB) platform.** Biffel and Schatten proposed a platform called Engineering Service Bus (EngSB) which integrates not only different tools and systems but also different steps in the software development lifecycle [5, 6]. Figure 1 shows an overview of the EngSB platform. The

<sup>1</sup> <http://www.automationml.org>

<sup>2</sup> <http://www.medeia.eu>

successful development of modern software-based systems, such as industrial automation systems, depends on the cooperation of several engineering disciplines, e.g., mechanical, electrical and software engineering, so-called (software+) engineering environments. The EngSB addresses requirements such as the capability to integrate a mix of user-centered tools and backend systems, mobile work stations that may go offline, and flexible and efficient configuration of new project environments and SE processes.

**Tool Domain concept.** The EngSB platform introduces the concept of tool types that provide interfaces for solving a common problem, independent of the specific tool instance used. This seems possible since different tools, developed to solve the same problem have, more or less, similar interfaces. For example, the source code management (SCM) tools *Subversion* and *CVS* both provide similar functionality, which allows describing these tools as instances of the SCM tool domain. Figure 2 illustrates the SCM tool domain and other possible domains in the context of the EngSB. We call the concept of tool types in this work “tool domains” [5]. This concept allows the EngSB to interact with a tool domain without knowing which specific tool instances are actually present. Note that tool domains do not implement tool instances but provide the abstract description of events and services, which have to be provided by concrete connectors of tool instances to the EngSB.

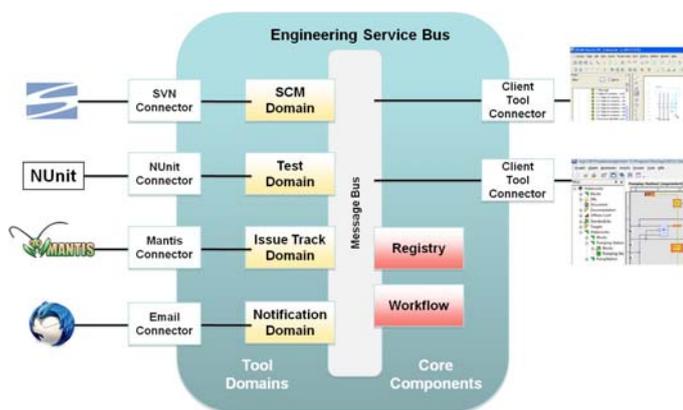


Figure 2. Tool Domain Concept of the EngSB.

### B. Model-driven System Configuration

The major goal of the Model Driven Architecture (MDA) approach is the separation of system functionality specification and implementation [11]. The advantages [10] of the MDA framework are (1) automated generation of results improving productivity, development duration, and cost; (2) focusing on the creation of conceptual models rather than on logical and technical details. In contrast to the MDA approach, the Model-driven System Configuration (MDSC) [5, 16] automatically derives integration technology configurations from business requirements rather than implementation code. Based on requirement and capability models which represent documents, integration expert knowledge and estimation/measurements of the integration network capabilities [12], a logical solution model which represents the set of suitable integration partners (i.e. business services) is derived automatically [13]. Based on described network capability models and the derived

integration partners the logical solution model is transformed into a technical solution model that represents the specific integration configuration for the underlying integration network technologies. The configuration specifies routing tables for efficient communication, backup routes for a fault-tolerant behavior, transformation instructions to enable data exchange between heterogeneous business services, and installation instructions for the deployment of middleware technologies to cope with heterogeneous network technologies.

In contrast to MDSC, the eMDSC approach focuses on specifying the proper communication sequence between tool instances based on a description of an engineering process. The configuration therefore focuses on the efficient integration of tool instances rather than on configuring the integration platform regarding efficient routing and effective fault-tolerance.

### III. RESEARCH ISSUES

Typical (software+) engineering processes involve a set of heterogeneous engineering domains and tools, which need to be integrated to allow seamless cooperation and engineering process support across domain- and tool-boundaries. However, the configuration of IT technologies suitable for addressing these integration challenges often requires manual and therefore time-consuming and error-prone configuration tasks.

In this paper, we propose an extended model-driven approach for the automated derivation of integration technology configurations based on explicit semantic models of engineering process requirements and engineering tool capabilities. Based on these semantic models, the proposed approach is capable of efficiently and effectively deriving integration technology configurations. In order to investigate the feasibility and applicability of the proposed extended model-driven approach, we derive the following research issues:

**RI-1: Efficient derivation of integration technology configurations.** Investigate to what extent an automated derivation of integration technology configurations is achievable without the need for manual intervention. Discuss the required additional effort needed before executing the automated derivation, such as the effort for modeling engineering process requirements and engineering tool capabilities. Evaluate the effort needed for the adaptation of existing integration technology configurations in case of changed engineering processes, such as the exchange of existing and the deployment of new engineering tools.

**RI-2: Effective management of defects in the process of configuring integration technologies.** Discuss the advantages and limitations of the proposed extended model-driven approach regarding the handling of defects in the process of configuring integration technologies. To what extent can the proposed approach transform the explicit engineering process requirement models into correct and valid integration technology configurations without significant sources of defects like manual interaction; better quality measurement and feedback on intermediate models during configuration derivation?

We discuss the extended model-driven approach based on a standard software engineering process. We show the feasibility of the proposed approach and discuss the advantages and limitations in the context of software(+) engineering. For empirical evaluation we determine the integration effort needed for each configuration process step to compare the steps in the proposed extended model-driven approach with both the traditional model-driven approach as well as the manual approach.

#### IV. USE CASE

This section describes the use case of the standard Continuous Integration and Test (CI&T) process. We selected the CI&T process approach because of the involvement of a set of various tools (build, automated tests, and deployment) as a representative best-practice approach from the agile software engineering. Nevertheless, this use case seems appropriate for illustrating the proposed eMDSC approach.

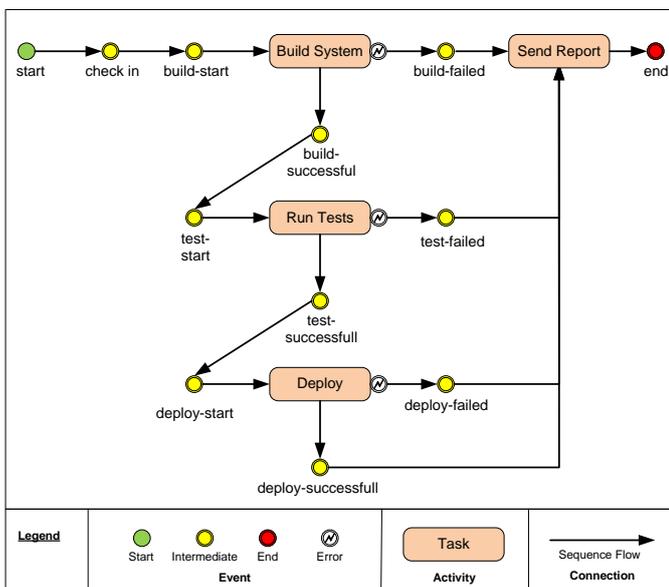


Figure 3. Continuous Integration and Test (CI&T) Process.

The expected SE process model for the CI&T use case is represented in Figure 3 using Business Process Modeling Notation (BPMN)<sup>3</sup> notation. The model consists of a set of activities for the CI&T process implementation: building the system, running tests, deploy activities, and finally reporting test and deployment results. The CI&T use case shows a key feature of an iterative software development process: if parts of a system or engineering model get changed, the system has to be rebuilt and tested in order to identify defects early and to provide fast feedback on the implementation progress to the project manager and the owners of the changed system parts. In modern SE environments this part is done by Continuous Integration (CI) servers like Continuum<sup>4</sup> or Hudson<sup>5</sup>. For a

typical Java project a Maven<sup>6</sup> or Ant<sup>7</sup> script will guide the CI process [5].

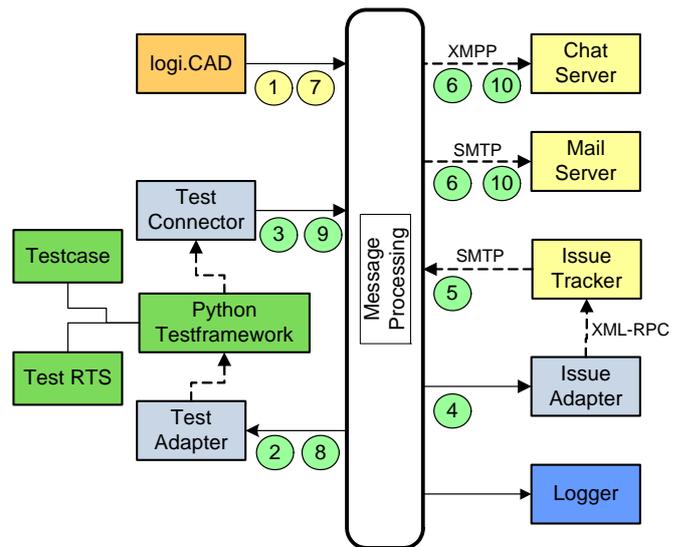


Figure 4. Overview and Sample Tools for CI&I Process.

As shown in Figure 4, a typical execution of the CI&T Process may be performed as follows. (1) an updated version of a software component is put into the archive using a client tool such as logi.CAD<sup>8</sup>. This leads to (2) the execution of the testing process, i.e., a service call to the test adapter. This test adapter then executes the test, e.g., using the shown Python test framework. After the tests are finished, the test results are put back into the Engineering Service Bus (EngSB) (3). If the tests resulted in one or more errors, for each of these errors an issue (e.g., an email or a ticket) is generated (4) using the issue adapter of the EngSB. The EngSB is then notified (5) about each of these newly created issues and (6) notifies all interested roles/persons, e.g., by using a chat server. After the errors of the originally checked-in component have been resolved, the component is checked-in again (7), again the tests are executed (8) and the, now error-free, test results are put back into the EngSB (9). Finally, all interested roles are notified (10) about the successful updated of the software component.

#### V. AUTOMATED DERIVATION OF CONFIGURATIONS

The automated derivation of configuration for the integration of tools across engineering boundaries is described in Figure 5. The process of configuration derivation consists of 4 main processing components and 4 QA steps described in the following.

<sup>3</sup> <http://www.bpmn.org>

<sup>4</sup> <http://continuum.apache.org>

<sup>5</sup> <http://hudson-ci.org>

<sup>6</sup> <http://maven.apache.org>

<sup>7</sup> <http://ant.apache.org>

<sup>8</sup> <http://www.logicals.com>

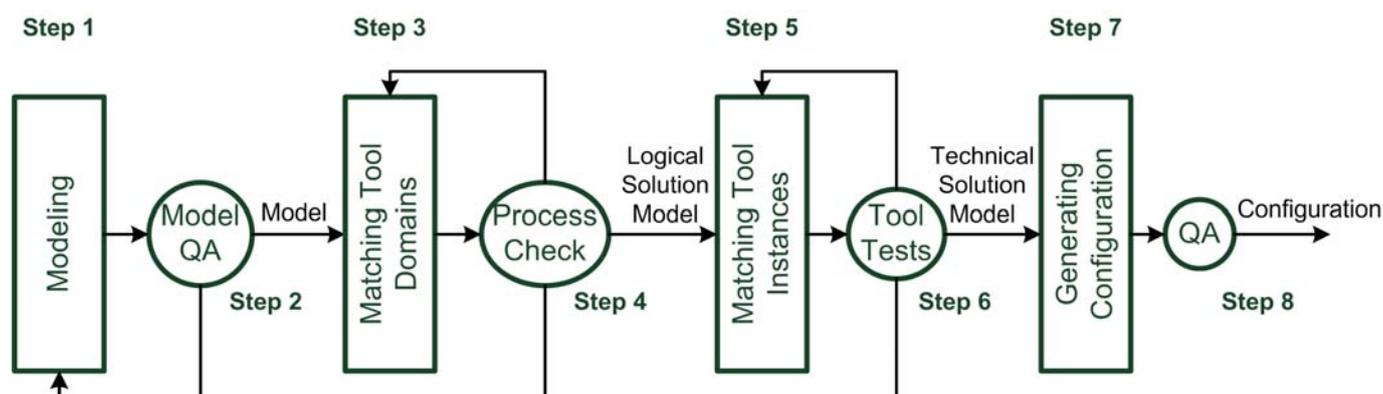


Figure 5: The process of configuration derivation for tool integration in (software+) engineering domains.

**Step 1.** In the first processing step engineering tool experts from each engineering domain independently model the capabilities of used tool instances (e.g., issue tracker) within their domain as well as tool domain requirements and capabilities. Furthermore, engineering process experts model the requirements of the engineering process requiring the integration of several engineering disciplines. A way of modeling capabilities and requirements based on ontologies is described in [17]. The result of this step is semantically described models.

**Step 2.** In the second processing step first QA steps are deployed to check models automatically for correctness and syntax validity [3, 4]. The QA step may make use of reasoning capabilities provided by e.g., protégé to assure certain model constraints (e.g., security properties). In case of errors models described in step 1 need to be refined.

**Step 3.** In the third processing step a matching between modeled engineering process requirements and modeled tool domain capabilities is performed. The approach of matching capabilities and requirements using ontologies is described in [18, 19]. The result of this step is a possible set of integration partners where engineering process requirements are matched by several tool domain capabilities. In such case engineering process experts need to perform a manual selection of suitable tool domains based on characteristics of the derived tool domains which were not explicitly used in the matching process or cannot be explicitly modeled, like the confidence of the experts into the tool instances within the tool domain. The outcome of this processing step, i.e. Logical Solution Model, is an engineering process with well defined and explicitly referred tool domains capable of satisfying the process' requirements.

**Step 4.** The following QA step is capable of simulating engineering processes using selected tool domains by means of pre-defined test scenarios. The simulation shows whether the engineering process is capable of working as originally intended. In this step tool domains emulate tool instance functionality as defined in the test scenario. In case of errors or an invalid engineering process either the selection process has to be restarted and new tool domains found, or tool domain capabilities and engineering process models have to be redefined in more detail.

**Step 5.** Once proper tool domains have been found and tested for the given engineering process, appropriate engineering tool instances need to be derived by matching tool domain requirements and tool instance capabilities [18, 19]. The process of matching capabilities and requirements is the same as in step 3. The result is a set of tool instances matching tool domain requirements. Tool domain requirements represent the capabilities the tool domain promises the engineering process. In case there are several tool instances matching the specified requirements a further selection of suitable tool instances has to be performed. Criteria for final selection may base on non-functional requirements which may also be optional. The result of this processing step, i.e. Technical Solution Model, is a set of tool instances each matching the requirements of the tool domain they belong to.

**Step 6.** The derived tool-instances are tested based on test scenarios pre-defined in tool domains. Such scenarios check proper functionalities of engineering tools and therefore consistency between real tool capabilities and tool capability models. In case of inconsistencies either bugs in the engineering tool have to be fixed or the model updated accordingly. In the latter case the process of configuration derivation has to be restarted since initial conditions have changed.

**Step 7.** In the next processing step the technical solution model, representing selected tool instances and tool domains, is transformed into real configuration parameters. The generated configuration is useable by the underlying integration technology and represents the original engineering process. The described process creates configuration solution for the proposed EngSB. A more general approach taking into account technology specific aspects is explained in [16].

**Step 8.** Finally, in the last QA processing step the generated configuration solution is evaluated by means of consistency and syntax checks.

## VI. EVALUATION

The evaluation of the E-MDSC approach was conducted by means of a standard software engineering process regarding a set of evaluation criteria. We compare the proposed E-MDSC with both the traditional MDSC as well as with a primarily manual way of configuration. We derived the used evaluation

criteria together with experts from our industry partner in the field of automation systems engineering. The evaluation is based on two scenarios. The first scenario determines the results based on an integration project from the scratch. The second scenario assumes that an initial integration project has been accomplished providing a first integration solution, but due to changing business requirements some system adaptations have to be performed, like the need to exchange of a tool instance. Table 1 summarizes the initial results of our evaluation with respect to the derived criteria.

**Configuration Knowledge.** The results of the evaluation show that the main difference between the two MDSC variants and a manual configuration approach is the type of sources used for the derivation of the configuration. On the one hand side, human-readable documents or other implicit knowledge is used by experts to manually generate the configuration. On the other hand side, this knowledge is externalized in machine-understandable models which enable and act as input for an automated derivation of configuration parameters.

**Initial Effort.** The initial effort (i.e., an integration scenario starting from the scratch) needed for the manual derivation of configurations is lower than in case of automated derivation. The reason for this is that in case of MDSC, documented knowledge still needs to be transferred into explicit and machine-understandable format (e.g., ontologies), while in case of manual derivation this step is done implicitly by experts. Furthermore, the eMDSC variant requires slightly more effort than the traditional MDSC variant, since tool domains need to be described additionally.

**Adaptation Effort.** In case of adaptation, the MDSC variants have proven to be more efficient than the manual approach, since once the knowledge has been externalized, it can be reused with little extra effort, while for the manual approach this knowledge exists implicitly only. In addition, both MDSC variants report errors or missing information immediately due to in-time consistency and completeness checks based on ontology reasoning. In case of manual derivation, documents may be changed with the risk that other related documents are not updated accordingly, resulting in inconsistent and therefore erroneous knowledge. The eMDSC variant requires less adaption than the traditional MDSC variant, since there is an additional separation between tool instances and tool domains. The tool domain concept of the eMDSC allows more efficient modeling of new or updated engineering tool instances by providing templates for the core functionalities of each used tool domain. Additionally, the separation into engineering processes, tool domains and tool instances allows experts to entirely focus on specific parts of the model, rather than taking into account the entire model (e.g., in case of new or updated tool instances, the process of selecting the appropriate instance is limited to the tool domain the tool instanced belong to).

**Table 1: Evaluation results of manual, MDSC, and eMDSC approach.**

	Manual	Traditional MDSC	eMDSC
<b>Configuration knowledge</b>	Configuration knowledge has to be derived from human-readable documents or implicitly known by integration experts	Configuration knowledge is externalized in a machine-readable ontology by integration experts	similar to traditional MDSC
<b>Initial effort</b>	medium	high	slightly higher
<b>Adaptation effort</b>	high	medium	low
<b>Duration</b>	high	low	medium - duration is longer due to additional abstraction (tool domains need also to be modeled)
<b>QA efficiency</b>	Low - Manual checks of documents and models needed	Medium - Automated ontology reasoning allows quickly locating inconsistent knowledge in the model	High – additionally to MDSC tool domains allow tool domain-specific integration tests of tool instances
<b>Model complexity</b>	High and distributed	Medium and centralized	Low and centralized
<b>Level of automation support</b>	Low - Exhaustive communication of engineering tool experts needed to clarify configuration	Medium - Semantic models of engineering process requirements and engineering tool capabilities with ontology-based reasoning allows the automated derivation of configuration	High – additionally to MDSC, already modeled engineering processes can be efficiently reused because of their explicit modeling in the eMDSC approach

**Duration.** The duration of the manual approach is higher due to error-prone mainly manual process steps resulting in additional efforts to discuss error sources and possible solutions. In case of describing systems, parallel processing is possible in both approaches. However, the MDSC processing steps are running mainly automated, while the manual approach is still human-driven resulting in time consuming and error-prone processing steps. Therefore, the duration depends strongly on the automation support. The eMDSC variant requires a slightly longer duration, since the tool domains need to be described before the actual tool instances can be modeled.

**QA Efficiency.** Since the manual approach focuses on manual validity checks, it is therefore more time consuming and error-prone. This also results in the fact that missing information is often detected probably only in later process steps. The MDSC variants use ontology-based reasoning. This allows performing consistency and completeness checks in-time automatically, resulting in a lower failure rate and in-time notification of experts about missing/incorrect information. Furthermore, based on tool domains the eMDSC variant is capable of additionally executing integration tests to check whether all published functionalities of tool domains are correctly supported by tool instances. The tool domain abstraction allows experts to define tests on the engineering process level, rather than on the engineering tool instance level. This allows testing the engineering process even before tool instances have been modeled.

**Model Complexity.** The complexity of the manual approach is high because the process requires the cooperation of experts from the different engineering disciplines, as well as because of heterogeneous and distributed knowledge sources (e.g., documents) which have to be kept consistent all the time. The traditional MDSC variant is less complex because the knowledge is kept centralized in an explicit format. Each expert is responsible for maintaining the model of his/her engineering tool instances without the need for cooperation with other experts. However, the links between the different engineering disciplines (i.e., the link between tool instances belonging to different engineering disciplines) have to be modeled and kept consistent by designated engineering process experts. The complexity of the eMDSC variant is even lower because of the distinction between tool domains and tool instances. This even allows experts of the individual engineering disciplines to separately model either engineering processes, tool domains and tool instances without requiring knowledge of the entire engineering discipline.

**Level of Automation Support.** Besides supporting the engineering tool experts while modeling engineering tool instances with consistency and completeness checks, the traditional MDSC variant automatically suggests an engineering process to be verified by the engineering process expert. In contrast the manual approach requires exhaustive communication of engineering tool experts and manual update of knowledge sources to derive a valid configuration. The proposed eMDSC variant has a higher level of automation support, since it enables the reuse of already modeled engineering processes, resulting in the appropriate tool domains to be selected automatically for the new (or adapted) engineering process.

## VII. DISCUSSION

This section discusses the proposed eMDSC approach, as well as the initial results of the evaluation with regard to the defined research issues.

**Efficient derivation of integration technology configurations.** The explicit and machine-understandable knowledge of the eMDSC approach, as well as the explicit distinction between engineering process, engineering tool domain and engineering tool instance allows a derivation of integration technology configurations with a high level of automation. This results in an efficient configuration of integration solutions for engineering processes across various engineering disciplines. The level of automation and modeling methodology enables effective modeling (i.e., modeling of either engineering processes, engineering tool domains or engineering tool instances) and supports the reuse of particular parts of engineering processes for new automation systems engineering projects with minimal adaptation effort. Although the process of deriving configurations from the described models is more complex than in the traditional MDSC variant, the cognitive complexity of modeling is lower for the particular experts because of the explicit distinction between engineering processes, engineering tool domains and engineering tool instances.

**Effective management of defects in the process of configuring integration technologies.** In contrast to the manual approach and the traditional MDSC variant, the eMDSC variant provides a higher level of QA support for the process of integration technology configuration derivation. This higher level of support results from the explicit and machine-understandable modeling of knowledge sources and the distinction between engineering processes, engineering tool domains and engineering tool instances. These two aspects allow automated consistency and completeness checks for all types of used models (i.e., engineering process requirements, tool domain requirements and capabilities, tool instance capabilities, logical solution model, and technical solution model). Furthermore, the tool domain concept enable experts predefining tests on an engineering process level, even before suitable engineering tool instances have been modeled or identified. Manual intervention is still necessary, however limited to the selection of engineering tool domains and engineering tool instances out of a set of suitable candidates. Nevertheless, this manual intervention does not introduce additional sources of defects, since only valid candidates can be chosen.

The evaluation scenarios supported the feasibility of the eMDSC approach and provided promising initial results. However, practical issues such as effort and defect rates for setting up and using eMDSC within larger-scale need to be explored in settings with industrial experts.

## VIII. CONCLUSION AND FURTHER WORK

(Software+) engineering, such as the engineering of power plants, typically involves the cooperation of a set of both technically and semantically heterogeneous engineering tools. The configuration of technologies enabling the integration of these engineering tools originating from various engineering

disciplines is usually a manual and therefore time-consuming and error-prone task.

In this paper, we introduced the so-called eMDSC (extended Model-Driven Systems Configuration) approach for the automate derivation of integration technology configurations based on explicit and machine-understandable models of engineering process requirements, engineering tool domain requirements and capabilities, as well as engineering tool instance capabilities. The eMDSC is based on the Model-Driven Architecture (MDA) paradigm and therefore enables an efficient, less complex, and less error-prone configuration derivation process.

We evaluated the proposed eMDSC approach by using a well-known software engineering process, the Continuous Integration & Test (CI&T) process. Major results of the evaluation were a) that the proposed approach has proven to be efficient and effective, regarding both the effort needed for setting up the integration environment as well as the number of error sources; and b) that tool domains enable easy adaptations of existing integration solutions by allowing the efficient exchange of similar tools without affecting the existing engineering process.

Future work will include investigation of the eMDSC approach in large-scale industry projects regarding practical issues such as effort and defect rates. In addition, the usability of the eMDSC approach will be evaluated in settings with industrial experts.

#### ACKNOWLEDGMENT

This work has been supported by the Christian Doppler Forschungsgesellschaft and the BMWFJ, Austria.

#### REFERENCES

- [1] L. Aldred, W. van der Aalst, and M. Dumas, "Understanding the Challenges in Getting Together: The Semantics of Decoupling in Middleware," *BPM Center, Eindhoven, The Netherlands*, 2006.
- [2] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web services: concepts, architectures and applications*, Springer Verlag, 2004.
- [3] S. Biffel, R. Mordinyi, T. Moser, and D. Wahyudin, "Ontology-supported quality assurance for component-based systems configuration," *Proceedings of the 6th international Workshop on Software Quality*, ACM, 2008, pp. 59--64.
- [4] S. Biffel, R. Mordinyi, and A. Schatten, "A Model-Driven Architecture Approach Using Explicit Stakeholder Quality Requirement Models for Building Dependable Information Systems," *Software Quality, 2007. WoSQ'07: ICSE Workshops 2007. Fifth International Workshop on*, 2007, pp. 6-6.
- [5] S. Biffel, and A. Schatten, "A Platform for Service-Oriented Integration of Software Engineering Environments," *Eight Conference on New Trends in Software Methodologies, Tools and Techniques (SoMeT 09)*, 2009, pp. 75 - 92.
- [6] S. Biffel, A. Schatten, and A. Zoitl, "Integration of Heterogeneous Engineering Environments for the Automation Systems Lifecycle," *Proc. IEEE Industrial Informatics (IndIn) Conference*, 2009, pp. 576 - 581.
- [7] D. Chappell, *Enterprise Service Bus*, O'Reilly Media, Inc., 2004.
- [8] A. Doan, N.F. Noy, and A.Y. Halevy, "Introduction to the special issue on semantic integration," *SIGMOD Rec.*, vol. 33, no. 4, 2004, pp. 11-13.
- [9] P. Duvall, S. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk*, Addison-Wesley, 2007.
- [10] A. Halevy, "Why your data won't mix," *Queue*, vol. 3, no. 8, 2005, pp. 50-58.
- [11] G. Hohpe, "Conversation Patterns," *Dagstuhl Workshop Report*, 2006.
- [12] G. Hohpe, and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley Longman Publishing Co., Inc., 2003.
- [13] IEEE, "IEEE Recommended Practice for CASE Tool Interconnection: Characterization of Interconnections," *IEEE Standard 1175.2-2006*, 2007, pp. c1-36.
- [14] Medeia-Consortium, "Requirements Analysis and Technology Review," *Medeia*, 2008.
- [15] S.J. Mellor, K. Scott, and D. Weise, *MDA distilled: principles of model-driven architecture*, Addison-Wesley Professional, 2004.
- [16] R. Mordinyi, T. Moser, E. Kühn, S. Biffel, and A. Mikula, "Foundations for a Model-Driven Integration of Business Services in a Safety-Critical Application Domain," *35th Euromicro Conference on Software Engineering and Advanced Applications*, IEEE, 2009, pp. 267-274.
- [17] T. Moser, R. Mordinyi, A. Mikula, and S. Biffel, "Making Expert Knowledge Explicit to Facilitate Tool Support for Integrating Complex Information Systems in the ATM Domain," *Intl. Conf. on Complex, Intelligent and Software Intensive Systems (CISIS 2009)*, IEEE, 2009, pp. 90-97.
- [18] T. Moser, R. Mordinyi, W.D. Sunindyo, and S. Biffel, "Semantic Service Matchmaking in the ATM Domain Considering Infrastructure Capability Constraints," *21st International Conference on Software Engineering and Knowledge Engineering (SEKE 2009)*, 2009, pp. 222-227.
- [19] T. Moser, K. Schimper, R. Mordinyi, and A. Anjomshoaa, "SAMOA - A Semi-Automated Ontology Alignment Method for Systems Integration in Safety-Critical Environments," *Complex, Intelligent and Software Intensive Systems, 2009. CISIS '09. International Conference on*, 2009, pp. 724-729.
- [20] N.F. Noy, A.H. Doan, and A.Y. Halevy, "Semantic Integration," *AI Magazine*, vol. 26, no. 1, 2005, pp. 7-10.
- [21] T. Rademakers, and J. Dirksen, "Open-source ESBs in action," *Manning Publications*, 2008.
- [22] R.M. Rangan, S.M. Rohde, R. Peak, B. Chadha, and P. Bliznakov, "Streamlining Product Lifecycle Processes: A Survey of Product Lifecycle Management Implementations, Directions, and Challenges," *Journal of Computing and Information Science in Engineering*, vol. 5, 2005, pp. 227-237.