Automated Tailoring of Application Lifecycle Management Systems to Existing Development Processes

Matthias Biehl, Jad El-khoury, and Martin Törngren
Embedded Control Systems
Royal Institute of Technology
Stockholm, Sweden
{*biehl, jad, martin*}*@md.kth.se*

***Abstract*—Application lifecycle management approaches are used to tame the increasing complexity, size and number of development artifacts. Throughout the application lifecycle, a number of tools are used to create a diversity of development artifacts. It is widely believed that the efficiency of development can be improved by the integration of these tools. However, such integrated solutions are not accepted by practitioners if the solutions are not aligned with the established development culture, processes and standards. Thus, application lifecycle management needs to be tailored to the specific corporate needs. The tailoring, however, is typically performed manually and is thus resource intensive. We propose a cost efficient tailoring approach for application lifecycle management, which is based on reuse and automation. We explore to what extent existing process models can be reused for automatically configuring the application lifecycle management system, so it is aligned with the development process. We identify a number of relationship patterns between the development process and its supporting tool chain and show how the patterns can be used for constructing a tool chain. In three case studies, we examine the practical applicability of the approach.**

Keywords-Application Lifecycle Management; Process Modeling; Tool Integration; Tool Chain; Generative Approach; Model Driven Development.

I. Introduction

The development of software-intensive products, such as embedded systems, produces a large number of diverse development artifacts, such as documents, models and source code. The artifacts are produced and used throughout the product lifecycle and are ideally managed systematically in an application lifecycle management (ALM) system [1]. In this article, we focus on one specific aspect of application lifecycle management systems – the aspect of tool integration. Currently available commercial application lifecycle management systems do not provide adequate tool integration, as shown in a recent analysis [2]. Tool integration is an essential aspect of ALM, since the development artifacts in the ALM system are typically developed with a number of different development tools. The development tools ideally interoperate seamlessly, however, the tools are often "island solutions" and a considerable engineering effort is necessary to make a specific set of tools interoperate. Thus, tool integration is realized externally to the development tools, in the form of tool chains. A *tool chain* can be regarded as an integrated development environment consisting of several development tools, which is intended to increase the efficiency of development by providing connections between the tools used in a development process [3].

To be effectively used, tool chains need to be customized to a specific selection of development tools and a specific development process. For example, a company might select IBM DOORS for requirements management, Enterprise Architect for UML modeling and MATLAB/Simulink for designing and simulating control algorithms. The tool chain needs to include these development tools. The way in which these tools are connected, is determined by the development process, which might prescribe that a connection between requirements and UML models is needed, and a connection between UML models and MATLAB/Simulink models.

Building automated tool chains that fit the individual needs is an expensive and time-consuming task. In Section II, we describe the challenges involved in building such customized tool chains and study the perspective of involved stakeholders. If a systematic and automated development approach for tool chains was available, tool chains could be efficiently developed for each new development context. We introduce a domain-specific modeling language for tool chains in Section III. The language allows us to express the essential design decisions for creating a tool chain. In Section V, we propose a systematic development process for building tool chains with this language, including the design phase, analysis phase, verification phase and implementation phase of tool chains. In Section IV, we focus in-depth on the relationship between process models and tool chain models and ways of leveraging this relationship in the conceptual design and verification phases of tool chain construction. We describe the relationship between process and tool chain in the form of patterns, implement them as model transformations and leverage these patterns for design and verification. We apply the approach in three case studies in Section VI. In the remaining Sections VII - IX, we relate our approach to other work in the field, sketch future work and consider the implications of this work.

II. Challenges

To develop modern software-intensive systems, such as an embedded system, a large number of development tools are used. Each of these tools can help us to be more productive, manage knowledge, and manage the complexity of development. The use of single, specialized tools has the potential to improve the efficiency of the development process, improve knowledge management, and improve complexity management, depending on the degree of automation they provide [4]. Multiple tools have the potential to improve the productivity in the development process, depending on how well they are integrated with each other and their degree of automation [3]. The reasons for using *multiple* tools can be found in the high degree of specialization of the tools, which is necessary to support the different engineering disciplines and the different engineering phases. The engineering of a software intensive system requires experts from a number of different engineering disciplines, such as control, hardware, software and mechanics. Each engineering discipline prefers a different set of development tools that excel in that particular discipline [5]. Throughout the different phases of the development process, specific tools are used, such as tools for prototyping, requirements engineering, design, implementation, verification and testing. In addition, crosscutting tools are used that support the process as a whole, such as repositories or tools for data management. The used tools are for the largest part commercial-off-the-shelf tools. The tools can thus not be changed and have to be used as they are.

Since engineers use the various tools to develop a single system, they need to relate the data that is captured in different tools, exchange data for reusing it in another tool or even to automate tasks that involve different tools. Most development tools do not interoperate well with one another, this is why additional software external to the tools – a tool chain – is needed as the glue to facilitate the integration.

Tool chains can provide different coverage of the development process; therefore, we distinguish between task-oriented tool chains with a small coverage and lifecycle-oriented tool chains with a larger coverage. Many existing tool chains cover only one task in the development process, e.g., the tool chain between source code editor, compiler and linker. We call these tool chains task-oriented. The tools are used in a linear chain, so that the output of one tool is the input for the next tool. These tool chains have a relatively small scope and integrate a small number of tools from within one phase in the lifecycle. Characteristic for these traditional tool chains are their linear connections, using a pipes and filter design pattern [6].

Along with the efforts to capture the complete application lifecycle in systems for ALM, there is a need for tool integration with a larger scope. Lifecycle-oriented tool chains support data exchange, tracing, and automation across the complete development lifecycle, from requirements engineering over verification, design and implementation to maintenance. When creating software-intensive systems, such tool chains may span multiple disciplines such as software engineering, hardware engineering and mechanical engineering and integrate a large number of different development and lifecycle management tools. In addition, modern development processes put new demands on the tool chain: processes might be agile, iterative or model-driven, which implies that the supporting tool chain cannot be linear.

With the large number of alternative development tools available in the marketplace and the large number of company-specific development processes, there is an even larger number of potential, different development processes that need to be supported by tool chains. A static, one-size-fits-all application lifecycle management system cannot fulfill these needs. Ideally, a tailor-made, customizable solution is available that addresses the individual needs. Since there is limited methodological and tool support and little reuse of tool chain parts, either one-size-fits-all tool chains are used despite their suboptimal support or customized tool chains are built, in a mostly manual way, which requires a tremendous development effort and investment.

In this article, we present one approach for solving this issue: by providing cost-efficient methods for building tool chains, the individual development of tailored tool chains becomes feasible. The approach manages to be cost-efficient by reuse of existing information and automation of development activities. The approach thus provides an opportunity to bring tailored tool chains within the reach of industrial application.

A. Stakeholders

As part of the description of challenges, some of the most important stakeholders of tools and tool chains are introduced, as depicted in Figure 1, including their roles as *users* or *creators* of tools and tool chains. The embedded systems developers work with multiple development tools and take on the role of the *users of tools*. In addition, the embedded systems developers take on the role of the *users of tool chains*, motivated by the expected efficiency gains in development provided by tool chains. The vendors of tools for embedded systems development take on the role of the *creators of tools*. IT infrastructure deploys the development tools. Process engineers have the big picture of the development process, which is hopefully consistent with the actual development practices of the embedded developers. Tool integration specialists are the only ones who know the integration technologies and conventions. A challenge is the effective communication between the stakeholders, as it requires a description of the different needs and possibilities of the stakeholders on the appropriate level of abstraction.

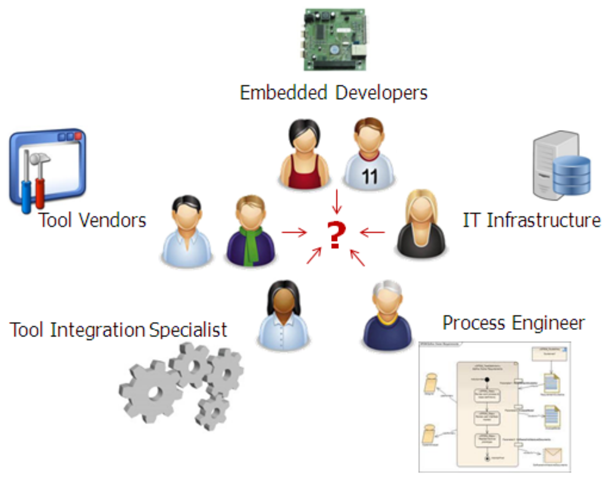

Figure 1. Stakeholders of tool chain development

Each one of the different stakeholders for tool chain development provides some important information for building a tool chain. Based on observations and interactions with industry in the research projects iFEST [7] and CESAR [8], the assignment of the role of the *creator of tool chains* is not clearly defined in industry. The role might be assigned to third party tool integration developers, but also to embedded systems developers or to tool vendors, which is problematic. Tool vendors are mostly interested in connecting only their tools to other tools, resulting in a limited scope of the integration. For embedded systems developers, the implementation of a tool chain is an additional burden that distracts them from their primary task of developing an embedded system. The observed constellation of stakeholders requires an approach for describing and communicating tool chains both in early design phases and in later phases, when more precision is needed.

III. Modeling the Design of Tool Chains

We need an early design model that describes all important design decisions of a tool chain. Such a design model can also serve as a boundary object [9] for the communication between different stakeholders. We chose to use the Tool Integration Language (TIL) [10], a domain specific modeling language for tool chains. TIL allows us not only to model a tool chain, but also to analyze it and generate code from it. The implementation of a tool chain can be partly synthesized from a TIL model, given that metamodels and model transformations are provided. Here we can only give a short overview of TIL, for an elaborated description of concrete graphical syntax, abstract syntax and semantics we refer to [10].

The graphical concrete syntax of each language concept is introduced by a simple example in Figure 2, the concrete mapping function, which maps abstract to concrete syntax, is defined by corresponding circled numbers ⓪..⑦ in Figure 2 and the following text. This section also briefly and informally introduces the semantics of TIL concepts.

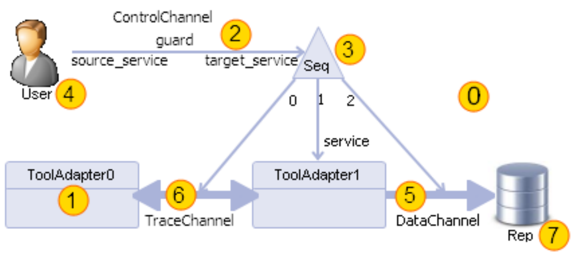

Figure 2. A simple TIL model illustrating the graphical concrete syntax of the language concepts

A **ToolChain** ⓪ provides a container for instances of TIL concepts. An instance of the ToolChain concept describes the tool chain by the composition of its contained instances of TIL concepts.

A **ToolAdapter** ① is a software component that describes the role of a tool in the tool chain by exposing the services and data of the tool, which are relevant for the specific role. Exposing the services of a tool enables control integration. Exposing the data of a tool enables data integration. A ToolAdapter makes two kinds of adaptation: (1) It adapts between the technical space of the tool and the technical space of integration for both data and services. (2) It adapts the structure of data and the signature of services available in the development tool to the data structure and service signatures defined by the ToolAdapter metamodel.

Each ToolAdapter has two associated *ToolAdapter metamodels*: one that specifies the structure of the exposed tool data and another that specifies the signature of the exposed services. In addition to the services defined in the metamodel, all ToolAdapters provide the default services *activate* to start the tool, *injectData* to load data (which is an instance of the ToolAdapter data metamodel) into the tool and *extractData* to access the tool data (as an instance of the ToolAdapter data metamodel). The ToolAdapter metamodels serve as an interface specification for the ToolAdapter and describe which data and services of the tool are exposed. More information on the structure of the ToolAdapter metamodels is provided in [10], [11].

Subtypes of ToolAdapters are defined, such as a **Repository** ⑦, which provides storage and version management, e.g., a ToolAdapter for Subversion [12].

A **DataChannel** ⑤ describes the possibility to transfer and transform data from a source ToolAdapter to a target ToolAdapter at the run-time of the tool chain; it is a directed connection. The data originates from the *source_service* of the source ToolAdapter (default service: *extractData*), is transformed and is finally received by the *target_service* of the target ToolAdapter (default service: *injectData*). A model transformation is attached to the DataChannel; the source and target metamodels of the transformation need to match the respective data metamodels of source and target ToolAdapters.

A **TraceChannel** ⑥ describes the possibility to establish trace links between the data of two ToolAdapters at the run-time of the tool chain; it is an undirected connection. A TraceChannel is a design-time representative for a number of trace links at run-time. At design-time one can specify the type of data that can be linked by traces. The endpoints of the traces can be restricted to a subset of the tool data by specifying the *source_service* and *target_service* (default service: *extractData*), which provide the data. At run-time, these services provide a list of all the source and target elements that are offered as endpoints for specifying a trace.

A **ControlChannel** ② describes an invocation or notification, it is a directed connection originating from a *source* component and ending in a *target* component. If the target of the ControlChannel is a ToolAdapter, the ControlChannel denotes the invocation of a tool service; if the target is a DataChannel, the data-transfer is executed; if the target is a TraceChannel, a dialog for creating traces is presented. If the target is a User, it denotes notification of the User. A condition for the execution of the ControlChannel can be specified by a guard expression. A service of the source component, called *source_service* (default value: *activate*), can be specified as the event that triggers the ControlChannel. The invoked service in the target component is specified as the *target_service* (default value: *activate*) of the ControlChannel.

A **Sequencer** ③ describes a sequence of invocations or notifications. When a Sequencer is activated by an incoming ControlChannel, it activates the outgoing ControlChannels in the specified order. The order is specified by the events (0..n), which are specified as the *source_service* in the outgoing ControlChannels from the Sequencer. Only after the service executed by the previous ControlChannel is finished, will the next ControlChannel be activated.

A **User** ④ represents a real-world tool chain user. The concept is used to describe the possible interactions of the real-world users with the tool chain. Outgoing ControlChannels from the User denote the invocation of tool chain services by the real-world user. Incoming ControlChannels to a User denote a notification sent to the real-world user, e.g., by e-mail.

By default, all TIL concepts describe parts of an automated tool chain, however some parts of the tool chain may not need to be automated and are manually integrated. TIL allows marking ControlChannels, DataChannels and TraceChannels as manually executed, in which case they are depicted by dashed lines.

The semantics of TIL is defined in the text above, in addition, compatible formal semantics of the behavior of TIL can be described by a mapping of TIL concepts to networks of finite state machines (FSMs) [13].

IV. Relationship between Process and Tool Chain

When building a tool chain, it is important to study which development tools need to be connected. This information about the relationship between development tools is often already available in a process model. Process models exist for a variety of reasons, i.e., for documenting, planning or tracking progress in a development project. The Software & Systems Process Engineering Metamodel (SPEM) [14] is the standardized formalism by the OMG for this purpose. A SPEM model might already be available independently from a tool integration effort, e.g., as it is the case in development with the Automotive Open Software Architecture (AUTOSAR) [15]. The information available in process models forms the skeleton of a tool chain, i.e., which tools are involved and how they are connected in the process. To construct an executable tool chain as a software solution, additional details are needed, e.g., information about the data of tools, how to access it, how to convert it and how to describe the relation between data of different tools. In this Section we evaluate, to what extent information from existing SPEM models can be used for constructing a tool chain.

Ideally, connections for all tools used throughout the development process are provided; and in this case the tool chain supports the development process. The process provides constraints and requirements for the construction of the tool chain. While generic process models are available, e.g., the SPEM models for the Rational Unified Process (RUP) [16] or for AUTOSAR [15], companies also create individual process models for various purposes, e.g., to customize these generic models to their individual environments, to document the development process, to plan the development process, to track the progress in the development or to document their selection of tools.

If both the process is described as a model and the tool chain is described as a model, information from the process model can be reused for constructing a tool chain model. This approach ensures that the tool chain and the process are aligned. Alignment decreases the friction experienced when using the development tools according to the process model. Process models only contain some, but not all information necessary for specifying tool chains. Especially the type of the connection between tools needs to be added later on.

1) Modeling the Product Development Process: In this section, we introduce a modeling language that is used for describing the development process. There are both formal and informal processes in companies, documented to different degrees and there is an increasing trend to model processes. Several established languages exist for modeling processes or workflows. These languages have various purposes, for example BPMN [17] and BPEL [18] describe business processes and SPEM describes development processes. We apply SPEM, since it is a standardized and relatively widespread language for modeling development processes with mature and diverse tool support. A SPEM model describes both the product development process and the set of tools used and can thus be applied to describe the process requirements of a tool chain. An example model is provided in Figure 5.

A number of concepts are defined in SPEM, we introduce here the core concepts that are relevant in the context of tool chains: a *Process* is composed of several *Activities*; an *Activity* is described by a set of linked *Tasks*, *WorkProducts* and *Roles*. A number of relationships, here represented by ≪.≫, are defined between the concepts of the metamodel: a *Role*, typically an engineer or software, can ≪*perform*≫ a *Task* and a *WorkProduct* can be marked as the ≪*input*≫ or ≪*output*≫ of a *Task*. A *WorkProduct* can be ≪*managed by*≫ a *Tool* and a *Task* can ≪*use*≫ a *Tool*.

A. Development Process Model as Requirements for Tool Chains

In general, a requirement is a documented need of the nature or behavior of a particular product or service. Requirements can have different degrees of formalization and structure. In the context of developing tool chains, the tool chain is the product. The nature or behavior of a particular tool chain is documented by process models, which thus can be interpreted as the requirements. Process models contain the selection of tools and the description of the connections between the tools. Since process models describe the process in a structured and formalized form, the requirements of a tool chain are formalized and structured, which we will use for describing the relationship between process and tool chain and using this relationship for efficiently constructing and verifying tool chains.

B. Relationship Patterns between Process and Tool Chain

If both the process and tool chain are described as a model, we can also model the relationship between them. A process described in SPEM might provide several opportunities for tool integration. Such an opportunity involves two tools and a direct or indirect connection between them. The tools and the connections found in SPEM are included into the tool chain architecture as ToolAdapters and Channels. The direction of the DataChannel can be determined by the involved work products, which have either the role of input or output of the task. Tasks connected to only one tool or tasks dealing with work products connected to the same tool do not require support from a tool chain; in these tasks engineers work directly with this tool, e.g., by using the GUI of the tool.

The challenge is to identify those parts in a SPEM model that are relevant for tool integration. The relationship cannot be described by mapping single metaclasses, as in Table I, instead the relationship needs to be described by combinations of several connected metaclasses, which we call patterns. To describe this relationship in more detail, we list patterns of both SPEM and TIL models and their correspondences.

Table I
CORRESPONDENCES BETWEEN SPEM AND TIL METACLASSES

SPEM Metaclass	TIL Metaclass
Role	User
Tool	ToolAdapter
Task	Channel

The relationship patterns consist of a SPEM part, which matches a subgraph of a process model in SPEM, and a TIL part, which will become a new subgraph in the tool chain model in TIL. The mapping is established by pairs of model elements from both SPEM and TIL, whose name attribute is equivalent and whose types are mapped according to the correspondences of metaclasses presented in Table I. In the following, we show four SPEM patterns that describe tool integration related activities, they are illustrated in the top part of Figure 3, and the corresponding TIL pattern is illustrated in the bottom.

1) *Task-centered Integration Pattern:* For each *Task* in SPEM that has associated *WorkProducts* as input and output, where the input *WorkProduct* has a different associated *Tool* than the output *WorkProduct*, this pattern produces *ToolAdapters* and a *Channel* between them in the TIL model. The pattern is shown in (1) and can be observed in case study 1 in Figure 5 for the *Task TraceReq2UML* connecting the *WorkProduct RequirementsDatabase* and the *WorkProduct UMLFile*.
2) *Multi-tool Task-centered Integration Pattern:* For each SPEM *Task* with two SPEM *Tools* associated with it, this pattern produces *ToolAdapters* and two *Channels* between them in the TIL model, since no directionality is modeled in SPEM. This pattern is theoretically possible according to the SPEM metamodel, but we have not observed it in practice. The pattern is illustrated in (2).
3) *WorkProduct-centered Integration Pattern:* For each SPEM *WorkProduct* that is both input and output of its associated *Tasks*, which have a different associated *Tool*, this pattern produces *ToolAdapters* and a *Channel* between them in the TIL model. The pattern is illustrated in (3) and can be observed in case study 2 in Figure 7 for the *WorkProduct ECUConfigurationDescription*, which is output of the *Task GenerateBaseECUConfiguration* and input to the *Task GenerateRTE*.
4) *Multi-tool WorkProduct-centered Integration Pattern:* For each SPEM *WorkProduct* in that is associated to two different *Tools*, this pattern produces *ToolAdapters*

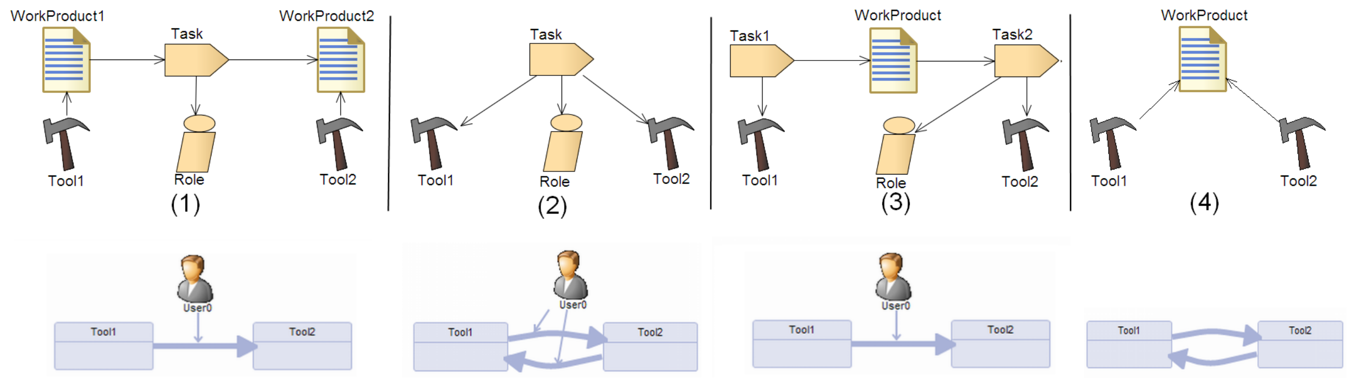

Figure 3. SPEM and TIL Patterns

and two *Channels* between them in the TIL model, since no directionality is modeled in SPEM. This pattern is theoretically possible according to the SPEM metamodel, but we have not observed it in practice. The pattern is illustrated in (4).

For all relationship patterns, the following constraints need to be fulfilled: For each *Role* in SPEM that is connected to the *Task*, we create a *User* model element in the TIL model, which means that the Role in SPEM and the User in TIL are optional parts in the patterns of Figure I. If a *ToolAdapter* corresponding to the *Tool* already exists in the TIL model, the existing *ToolAdapter* is connected, otherwise a new *ToolAdapter* is produced.

C. Implementation as Model Transformations

The implementation of the patterns offers possibilities for automation of the pattern usage. We implement the relationship patterns as model transformations, with SPEM as the source metamodel and TIL as the target metamodel. We chose the model-to-model transformation language in QVT-R, with the mediniQVT engine, and the Eclipse Modeling Framework (EMF) for realizing the metamodels. We use a simplified SPEM metamodel in EMF, and for the visualization of SPEM models we use Enterprise Architect. For modeling and visualization of TIL, we use the TIL Workbench described in [10].

Patterns (1) to (4) in Figure 3 are graphical representations of the relational QVT model transformation rules. Since QVT relational is a declarative language, the implementation describes the source patterns and the corresponding target patterns in the form of rules. Additionally, the attributes between source and target pattern are mapped, as described in Table I.

D. Usage of Relationship Patterns

The relationship patterns can be used in different ways. Here, we apply the relationship patterns for constructing the initial design of a new tool chain starting from a process model. Other forms of using the relationship patterns are possible, but are not considered in depth here. We can use the patterns, e.g., for verification: based on a process model and a tool chain model we check if the requirements provided by the process are realized by the tool chain model.

In the following, we focus on the application of the relationship patterns to create an initial tool chain design in TIL based on the process requirements expressed in the SPEM model. The patterns can be applied to a SPEM model that is complete and contains all necessary references to *Tools*. The patterns ensure that the design of the tool chain is aligned with the process, a necessity for acceptance of the tool chain with practitioners. This design of the tool chain can be created in an automated way and might need to be iteratively refined by adding details.

The process model only provides the skeleton for the specification of a tool chain, such as the selection of tools, the connections between the tools and the user role working with the tools. The process model does not provide the nature of the connections and the exact execution semantics of the automated tool chain. The nature of the connection can be data exchange, for creating trace links between tool data or for accessing specific functionality of the tool. This information needs to be added manually by configuring and choosing the right type of channel in TIL, a DataChannel, TraceChannel or ControlChannnel. Also, events need to be specified that trigger the data transfer or activate the tracing tool. For each ToolAdapter, a metamodel describing the data and functionality of the tool need to be added to the TIL model. For each DataChannel, a model transformation needs to be added.

To handle these cases, we add a refinement step, which complements the automated construction. Once this information is added, the TIL model can be used as input to a code generator for tool chains, as detailed in [11].

After focusing on the initial conceptual design phase for tool chains in this section, we explain the complete development process for tool chains in the following section.

V. Tool Chain Development Process

When developing tool chains, two processes are relevant: (1) the process for developing the embedded system as a product, called PDP (Product Development Process) and (2) the process for developing a tool chain (TCDP). The TCDP is followed at design-time of the tool chain to ensure that the developed tool chain can support the PDP at run-time by automating its integration-related tasks. In this section we address the TCDP.

A. *Overview*

In Figure 4, the TCDP – the process for developing a tool chain with TIL – is illustrated using the SPEM [14] notation. The development process for tool chains with TIL is structured into five phases: requirements engineering, conceptual design, detailed design, analysis and implementation. These phases are presented according to the order in which they are traversed during tool chain development. The complete tool chain development process has the following phases:

1) The requirements of the tool chain are elicited from the selection of tools and from the dependencies of tasks and tool usages in the product development process.
2) In the conceptual design phase, a conceptual model of the tool chain is described using TIL based on the requirements stipulated by the product development process. The conceptual model conveys the overall architecture of the tool chain, including the existing ToolAdapters, Users and connections between the ToolAdapters.
3) The alignment of the conceptual TIL model with the process model can be verified to highlight any intended or unintended discrepancies between tool chain design and the requirements stipulated by the product development process. Depending on the outcome of the analysis, the conceptual design phase can be iterated in order to create a conceptual model which is better aligned with the requirements.
4) In the detailed design phase, the conceptual TIL model is refined by different types of Channels and ToolAdapter metamodels. ToolAdapter metamodels are attached to each ToolAdapter in the TIL model to describe the data and services of the tool, which are exposed by the ToolAdapter. The connections between ToolAdapters and other components are refined by choosing the type of the connector (ControlChannel, DataChannel or TraceChannel). A model transformation is attached to each DataChannel in the TIL model; it describes the translation of data from the source tool to the target tool. The model transformation can be specified in different ways, either manually or computed based on the information in an ontology or weaving model). The conceptual TIL model with attached metamodels and model transformations yields a complete TIL model.

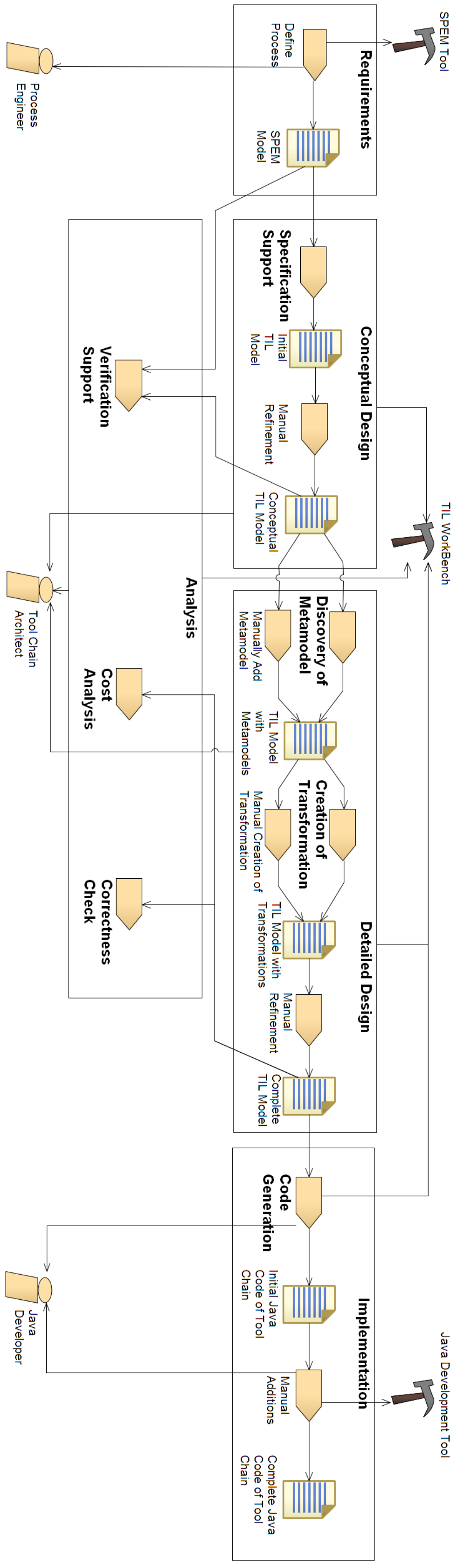

Figure 4. Process for developing a tailored tool chain with semi-automated support based on TIL, illustrated using SPEM [14] notation

5) Additional analyses are possible based on the detailed design. The syntactic correctness of the model can be checked and non-functional properties, such as the development cost of the tool chain, can be estimated. Depending on the outcome of the analyses, the tool chain design can be corrected before proceeding to the implementation phase.
6) In the implementation phase, the TIL model can serve as a blueprint for implementing the tool chain. The code of the tool chain is compiled and deployed.
7) At run-time[1] of the tool chain, the embedded developers use the deployed tool chain, which integrates several embedded systems tools.

There are different stakeholders of the tool chain, who are in contact with the tool chain at different points in the lifecycle of the tool chain. For this purpose, a distinction is made between the design-time of a tool chain and the run-time.

At design-time of the tool chain, the tool chain development process is executed, which involves a process engineer, tool chain architect and tool chain developers. A *process engineer* may model the product development process that is supported by the tool chain. The *tool chain architect* defines the conceptual and detailed design of the tool chain. One or several *tool chain developers* implement the tool chain as software based on the tool chain design.

At run-time, the tool chain software is executed to realize the data-transfer, traceability, invocation and notification to support the *product development process* of the embedded system. The *embedded systems developers* have the role of *tool chain users*.

The role of the *tool chain architect* has been explicitly introduced to cover the responsibility of specifying, refining and analyzing the architecture of the tool chain. Since TIL allows the tool chain to be described independently of implementation technology, the role of the tool chain architect can be separated from that of the tool chain developer. As the tool chain users, embedded systems developers are familiar with the requirements for the tool chain, but not with their implementation. Thus, embedded systems developers may be suitable candidates to take on the role of the tool chain architect and leave the implementation to dedicated tool chain developers. This separation of responsibilities is one attempt to resolve the unclear responsibility for the creation of tool chains observed in industry (cf. Section II-A).

B. Requirements Engineering

As described in Section IV-A, the product development process constitutes an important part of the requirements for the tool chain. By modeling the product development process in SPEM, we capture and model the requirements of the tool chain. We are thus in the situation to have semi-formal requirements for the tool chain.

[1]Strictly speaking, the run-time is outside the scope of the development process but has been added here for illustrating the connection between the tool chain and the embedded systems developers.

C. Conceptual Design

In the early design phase of the tool chain development process, a conceptual TIL model is created, which only describes the components and connections of the tool chain without any additional details. The conceptual model of the tool chain should be aligned to the product development process and the choice of development tools, so the tool chain can support the integration-related tasks in the development process. Development processes can be modeled for different purposes [19], however, here we focus on process models that have been created as a means for documentation, and are expressed by the Software and Process Engineering Metamodel (SPEM) [14].

If in addition to describing the tool chain as a model (e.g., using TIL), the process is also modeled (e.g., using SPEM), the relationship between the process model and tool chain model can be described. Possible relationships between SPEM and TIL models are identified and expressed by a mapping of a pattern of SPEM metaclasses to a pattern of TIL metaclasses. This mapping is implemented as a model transformation. Using the transformation and an existing process model, an initial conceptual design model of the tool chain is created. The main benefit of an automated mapping between process models and tool chain models is that an alignment between the design of the tool chain and the process can be achieved. This approach was described in detail in Section IV.

D. Refinement and Detailed Design

The conceptual TIL model needs to be refined by adding ToolAdapter metamodels that describe the data and the services exposed by each ToolAdapter and thus serve as interface specifications. If the ToolAdapter is to be newly implemented, the ToolAdapter metamodels need to be manually specified.

If an existing, already deployed ToolAdapter is to be reused and integrated into a tool chain, such as an existing ToolAdapter provided by a third party, the integration of the ToolAdapter would be possible on implementation level. In this approach, however, we explore the *integration at model level*, since a complete model of the tool chain enables correctness checks, analysis of the tool chain and complete synthesis of the implementation. The integration on model-level entails representing the interface of the remotely deployed ToolAdapter by ToolAdapter metamodels. This work explores, how the ToolAdapter metamodels of the remotely deployed ToolAdapter can be automatically discovered and integrated into a comprehensive TIL model of the tool chain.

The approach allows for the efficient reuse of deployed ToolAdapters in a new tool chain, ensures the consistency between the ToolAdapter metamodel and the deployed implementation, and the consistency between the ToolAdapter metamodel and the TIL model, enabled by the representation of all relevant information on the model level. This approach is further detailed in [20].

The conceptual TIL model needs to be further refined with detailed specifications for each DataChannel. DataChannels denote the transfer of data from a source ToolAdapter to a target ToolAdapter. The tool data is exposed by the ToolAdapter in the form of a model that conforms to the ToolAdapter metamodel. If the metamodels of source and target ToolAdapters are the same, the data can be simply copied between the ToolAdapters. In the more common case that the metamodels are different, the data needs to be transformed before it can be transferred to the target ToolAdapter. For this purpose, TIL offers the possibility to link a model transformation to each DataChannel. The model transformation converts the data between the metamodels of source and target ToolAdapters.

Typically, the details of a DataChannel are manually specified in the form of a model transformation, which requires time and effort. Especially if the requirements for the tool chain are still changing and prototypes of a tool chain are developed, an automated approach for the specification of model transformations can be valuable. In this setting, the intention is to rapidly and automatically create a first prototype of a model transformation, which can be manually refined later on.

Under certain conditions it might be possible to provide support for specifying a prototype model transformation automatically. The TIL model contains relevant information for generating the transformation, such as its execution direction and both its source and target metamodels. This information is not sufficient for an algorithmic approach, but a heuristic approach for prototyping model transformations can be realized. With the assumption that similar metaclasses of the metamodels of source and target ToolAdapters should be mapped, a model transformation can be computed using heuristics. As a measure for the similarity of the metaclasses, the similarity of the reference structure and the names of the metaclasses are used. The automatic refinement of the tool chain model by generating the information in DataChannels is described in [21].

E. Verification and Analysis

The analysis of a tool chain design is intended to support the *tool chain architect* when designing a tool chain. An advantage of using an explicit model-based description of the tool chain is the possibility for early analysis. Early analysis allows for evaluating different designs of the tool chain and especially allows finding problems during design that would be more expensive to correct if discovered later [22]. Instead of applying generic, existing analysis techniques, we here focus on *domain-specific analyses* that make use of the additional knowledge of the domain of tool integration, which is encoded in TIL models.

1) Correctness Checks of the Tool Chain Design: Correctness checks are used to detect specification errors in TIL models. Syntactic correctness of the TIL model is checked by the TIL Workbench when a TIL model is created. In addition, the following checks for semantic correctness are performed. A TIL model provides language concepts for both specifying service signatures and invoking the services. All service calls need to be consistent with their respective specification. Correctness checks compare the usage with the specification. The services and data structures are specified in the ToolAdapter metamodels and are used in the ControlChannels. The TIL model is checked for correctness by analyzing whether all service usages in the ControlChannels comply with their definitions in the ToolAdapter metamodels. The correctness check in the current implementation checks whether the used services are defined in the ToolAdapter metamodels by using the name of the services. Future work on the implementation could also take the parameters of the services into account.

2) Early Structural Design Verification of Tool Chain Design: In general, design verification checks if the design fulfills the specified requirements. The requirements for a tool chain are provided by the selection of tools, the product development process and additional information. Here the verification effort focuses on *structural design verification*, which is concerned with the extent to which the structure of the design of the tool chain is aligned to the structure required by the product development process. Early verification of tool chain design can detect possible misalignments between the structure of the product development process and the structure of the tool chain, when corrections are still relatively simple and cheap. By automating the early verification of tool chain design, it can be performed repeatedly with little effort, supporting the iterative refinement of tool chains.

The alignment of the design provided by a TIL model is checked against the requirements provided by a SPEM model [23]. This alignment can be expressed by a mapping of a pattern of SPEM metaclasses to a pattern of TIL metaclasses. Even if the conceptual model has been constructed based on a SPEM model, unintended changes might have been introduced by manual refinements. The verification produces a list of misalignments and a measurement indicating the degree of alignment between the tool chain and the product development process using precision/recall metrics [24], where a tool chain that is well-aligned to the process model has both a high degree of precision and a high degree of recall.

Structural design verification is only one part of a comprehensive design verification, since other requirements besides the structure, such as the behavior and non-functional requirements, need to be checked as well. Even a comprehensive design verification is a complement – not a replacement – to testing and verification of the final implementation.

F. Implementation

To support the implementation phase of the tool chain development process, the TIL approach provides a code generator. For any correct TIL model the code generator synthesizes a corresponding implementation automatically. TIL is designed to be independent of any particular implementation technology and thus code for different implementation technologies could be generated for a TIL model. For the purpose of showing that code generation is feasible, a particular implementation technology was chosen as a target platform and a code generator was built for it.

Code generation can produce a large part of the implementation automatically, however, it needs to be complemented with some manual implementation for interfacing the APIs of the integrated tools. We refer to [10] and [11] for a detailed description of the support for the implementation phase and code generator.

VI. Case Studies

In this section, we apply the identified relationship patterns between a process model and a tool chain (see Section IV) in two industrial case studies and a case study that recursively applies the approach to the development of tool chains. The tool chain development process (see Section V) and the tool integration language (see Section III) are used as enabling technologies. The variety of case studies gives us the opportunity to study different ways of using the patterns and to explore the impact of different modeling styles.

A. Case Study 1: Conceptual Design of a Tool Chain Model for a Hardware-Software Co-Design Process

This case-study deals with an industrial development process of an embedded system that is characterized by tightly coupled hardware and software components. The development process for hardware-software co-design is textually described in the following:

- The requirements of the embedded system are captured in the IRQA[2] tool. The system architect designs a UML component diagram and creates trace links between UML components and the requirements.
- The UML model is refined and a fault tree analysis is performed by the safety engineer. When the results are satisfactory, the control engineer creates a Simulink[3] model for simulation and partitions the functionality for realization in software and hardware.
- The application engineer uses Simulink to generate C code, which is refined in the WindRiver[4] tool. The data from UML and Simulink is input to the IEC-61131-3 conform ControlBuilder tool. The data from ControlBuilder, Simulink and WindRiver is integrated in the Freescale development tool for compiling and linking to a binary for the target hardware.
- A hardware engineer generates code in the hardware description language VHDL from Simulink and refines it in the Xilinx ISE[5].

[2]http://www.visuresolutions.com/irqa-web
[3]http://www.mathworks.com/products/simulink
[4]http://www.windriver.com

Based on the description of the process, we have created the corresponding SPEM model visualized in Figure 5.

We apply the model-to-model transformation that realizes the relationship patterns on the SPEM model in Figure 5. This yields a tool chain model that is aligned with the process, as shown in Figure 6. By applying the task-centered integration pattern shown in (1), we identify integration tasks that are linked to two work products that in turn are linked to different development tools (e.g., the task *Trafo_UML2Safety*). Some other tasks are not concerned with integration, they are related to one tool only (e.g., the task *Use_UML*).

The TIL model resulting from application of the relationship patterns is internally consistent; this means that there are no conflicts, missing elements or duplications in the model. All tools mentioned in the SPEM model are also present in the TIL model as ToolAdapters and all ToolAdapters are connected. In addition, the approach ensures that the design of the tool chain matches the process.

Since the tool chain is modeled, we can easily change, extend and refine the initial model before any source code for the tool chain is developed. The TIL model is relatively small compared to the SPEM model, thus hinting at its effect to reduce complexity. When using the simple complexity metric of merely counting model elements and connections, we see that in the TIL model their number is reduced by $2/3$ compared to the SPEM model (cf. Table II).

Table II
Size of the SPEM and TIL Model of Case Study 1

Count	Model Elements	Connections
SPEM Model	43	71
TIL Model	13	26

The important architectural design decisions of the tool chain (such as the adapters and their connections) can be expressed in TIL, while the complexity has been decreased compared to a SPEM model (cf. Table II). The tool chain model can be analyzed and - after additional refinement with tool adapter metamodels and transformations - can be used for code generation, as detailed in [11], [10]. Moreover, the presented model-driven construction of the tool chain ensures that the tool chain is aligned with the process.

[5]http://www.xilinx.com/ise

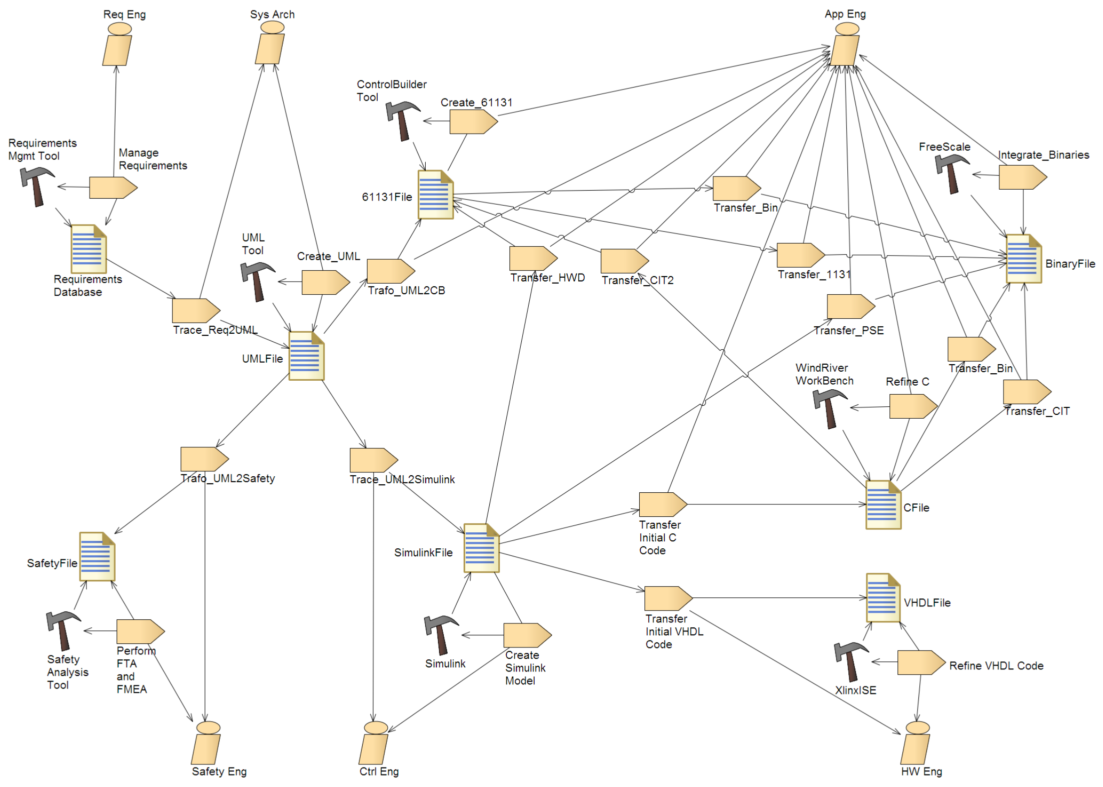

Figure 5. Case Study 1: Product Development Process of the Case Study as a SPEM Model

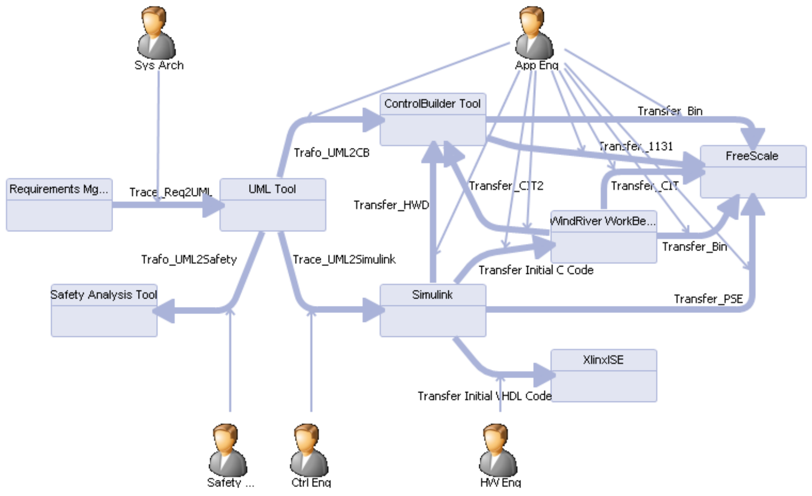

Figure 6. Case Study 1: Tool Chain of the Case Study as a TIL Model

B. Case Study 2: Verification of a Tool Chain Model for AUTOSAR ECU Design

In this case study, we model a tool chain for AUTOSAR. AUTOSAR is developed by the automotive industry and defines an architectural concept, a middleware and in addition a methodology for creating products with AUTOSAR. The AUTOSAR methodology describes process fragments, so called capability patterns in SPEM. Generic AUTOSAR tool chains are implemented in both commercial tools and open frameworks, however, it is a challenge to set up tool chains consisting of tools from different vendors [25] and tool chains customized to the needs of a particular organization.

The SPEM process model is provided by the AUTOSAR consortium and is publicly available, which contributes to the transparency of this case study. An excerpt of this model that is relevant for the design of a ECU, is depicted in Figure 7. We use this excerpt of the SPEM model to initialize a tool chain. Applying the patterns creates the tool chain model in TIL, illustrated in Figure 8. Out of the four different SPEM parts of the relationship patterns (1) - (4), only the workproduct-centered integration pattern (3) matched several times in the SPEM model. This is due to the modeling style used in the AUTOSAR methodology, where *WorkProducts* are used as an interface for integrating tools.

The generated skeleton of the tool chain lays the foundation for ensuring that the AUTOSAR methodology can be realized by this tool chain. The skeleton can now be refined with metamodels, model transformations and the behavior.

C. Case Study 3: A Tool Chain for Developing Tool Chains

To create a tool chain for developing tool chains, we apply the approach recursively onto itself: the tool chain is the product that will be developed according to the process for developing the tool chains. Using the terminology introduced in Section V, this process is called tool chain development process (TCDP) illustrated in Figure 4. We now interpret the TCDP as the PDP and thus as the basis for tool chain creation: the TCDP is interpreted as a description of the requirements of the tool chain.

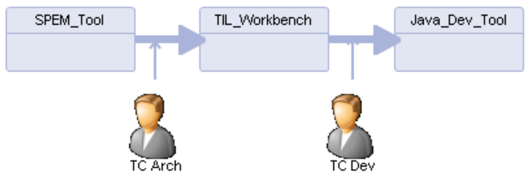

Figure 9. The tool chain for developing tool chains.

Only three tools are involved in the TCDP, and the tool chain is a straightforward pipe-and-filter architecture. The tool chain in TIL, which results from applying the patterns described in Section IV-B, is depicted in Figure 9. For each ToolAdapter, the metamodels are already defined and can be reused: The SPEM metamodel is defined [14] the TIL metamodel is defined [10] and the metamodel for Java is simply text in this context. The model transformations that need to be associated with the DataChannels are also defined: the model transformation from SPEM to TIL is described in Section IV-B in the form of patterns, the model transformation from TIL to Java code is described in paper [10]. With these cornerstones, it is thus perfectly feasible to apply the approach onto itself. This exercise can also be seen as an additional form of validation for the approach.

VII. Related Work

Related work can be found in the areas tool integration and process modeling. There are a number of approaches for tool integration, as documented in the annotated bibliographies [26], [27]. Most of the approaches do not take the process into account; in this section, we focus on those approaches that do.

Different process metamodels have been compared in [28] and specifically process models based on UML [29]. These approaches are usually focused on the description and documentation of processes. The execution of process models can range from simple workflow systems to more elaborate automation models. An example of an approach in the latter category is the PM+FDT approach [30]. It is based on a formalism for activity diagrams and uses model transformations to realize activities for transitioning from one formalism to another one. While dealing with multiple formalisms through transformations is possible, the connection to industrially used development tools is out of scope.

In the following, we classify approaches for process modeling according to two dimensions: The first dimension comprises different execution mechanisms, which can be interpretation vs. compilation. The second dimension comprises different process modeling languages, which can be proprietary vs. standardized.

Interpretation-based approaches [31], [32], [33] use the process definition for tool integration. This technique is also known as enactment of process models. Since the description of the process is identical to the specification of the tool chain, no misalignment between process and tool chain is possible. There are two preconditions for this approach: the process model needs to be executable and the access to data and functionality of the development tools needs to be possible. The use of a proprietary process model for interpretation in tool chains is introduced in [34], as the process-flow pattern. Approaches that extend SPEM make the process model executable [31], [32]. The orchestration of tools by a process model is shown in [33]. However, the interpretation of integration related tasks is often not possible, since the interfaces to the development tools are not standardized. Thus, the use of process enactment to build tool chains is limited.

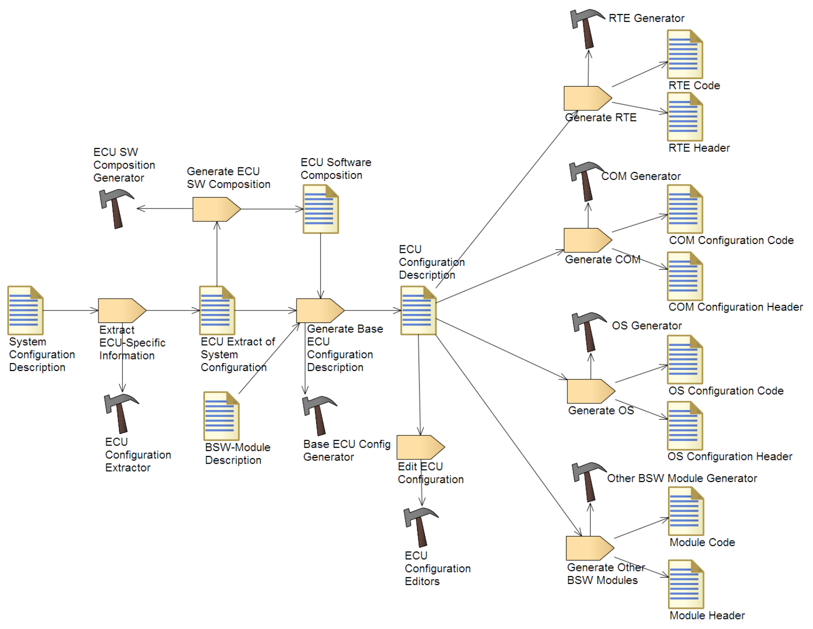
RTE Generator
RTE Code
Generate RTE
RTE Header
ECU SW Composition Generator
Generate ECU SW Composition
ECU Software Composition
COM Generator
COM Configuration Code
Generate COM
COM Configuration Header
ECU Configuration Description
System Configuration Description
Extract ECU-Specific Information
ECU Extract of System Configuration
Generate Base ECU Configuration Description
OS Generator
OS Configuration Code
Generate OS
OS Configuration Header
ECU Configuration Extractor
BSW-Module Description
Base ECU Config Generator
Edit ECU Configuration
Other BSW Module Generator
Module Code
ECU Configuration Editors
Generate Other BSW Modules
Module Header

Figure 7. Case Study 2: Excerpt of the AUTOSAR Methodology for Designing an ECU [15].

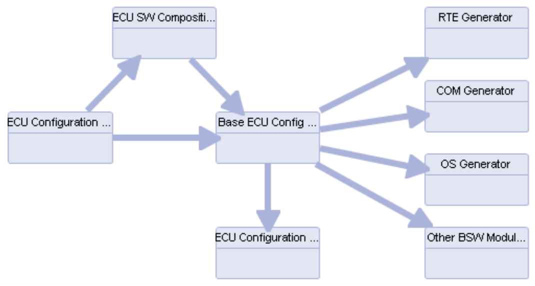

Figure 8. Case Study 2: AUTOSAR Tool Chain for Designing an ECU as a TIL Model

Compilation-based approaches transform the process model into another format, where the process model serves as a set of requirements. Proprietary process models provide great flexibility to adapt them to the specific needs of tool integration. An integration process model is developed in [35], where each process step can be linked to a dedicated activity in a tool. For execution, it is compiled into a low-level process model. The proprietary process model needs to be created specifically for constructing a tool chain. In this work, we use the standardized process metamodel SPEM [14], which allows us to reuse existing process models as a starting point for building tool chains and as a reference for verification for tool chains.

VIII. Future Work

This approach assumes that an appropriate process model for tool chains is available. We assume that the process model does not contain any integration related overhead, i.e., explicit representation of a model transformation tool and intermediate data model. We assume that tools have been assigned to process activities. The choice for certain tools is usually independent of automating the tool chain, the choice merely needs to be documented in the process model. SPEM offers several ways for describing tool integration. Future work can identify additional SPEM patterns for describing tool integration. Future work can also identify possible uses of additional SPEM constructs for describing tool integration, such as the the SPEM work breakdown structure.

The use of the presented patterns is limited to processes represented in SPEM and tool chains modeled in TIL. However, the patterns could be adapted to similar process metamodels, as long as the required concepts are present. In the future, we would like to experiment with additional languages for describing the process model, such as BPMN. This might help us to further generalize the patterns.

We have evaluated the approach in two case studies from the area of embedded systems and one case study from software development. We do not see any reason why the patterns could not be applied for creating tool chain from process models in other application areas in the future and are thus generalizable. For further validation, we thus plan to apply the presented techniques in another area of software and systems engineering.

IX. Conclusion

In modern development processes, tools are no longer used in a linear sequence, but as networks of interacting tools. The tool chain represents this network of interacting tools that needs to be tailored to the the development process. Processes are increasingly described as process models, which exist for a variety of reasons, i.e., for documenting, planning or tracking progress in a development project. SPEM is the standardized language by the OMG for this purpose. In this article, we recognize the development process modeled in SPEM as a set of requirements for the architecture of tool chains. We devise a number of patterns for creating the skeleton of a tool chain model in TIL, which is aligned with the process. This allows us to automate the creation of an initial tool chain design.

We have further shown how a tool chain model can be systematically developed into a tool chain implementation by following a structured tool chain development process. We have shown that many phases of this tool chain development process can be automated. If this process is followed, a semi-automated construction of tool chain software is possible. The semi-automation makes the construction cost-efficient, which is one of the decisive factors for building and configuring tailored application lifecycle management systems.

The presented cost-efficient construction has the potential to resolves the dilemma faced by industry today: Application lifecycle management can only deliver the promised efficiency improvements and cost savings, if it is tailored to the given set of tools and processes, but tailoring itself is expensive, has a cost and thus reduces the net-value of application lifecycle management. By automating the tailoring, as described in this article, the cost of tailoring is reduced and thus the net-value of application lifecycle management is highly improved.

Acknowledgement

The authors acknowledge partial funding by the ARTEMIS project iFEST.

References

[1] M. Biehl and M. Törngren, "Constructing Tool Chains based on SPEM Process Models," in *Seventh International Conference on Software Engineering Advances (ICSEA2012)*, Nov. 2012.

[2] C. Singh and M. Azoff, "Ovum Decision Matrix: Selecting an Application Lifecycle Management Solution, 2013-14," Ovum, Tech. Rep., Jan. 2013.

[3] M. Wicks and R. Dewar, "A new research agenda for tool integration," *Journal of Systems and Software*, vol. 80, no. 9, pp. 1569–1585, Sep. 2007. [Online]. Last accessed June 2013. Available: http://dx.doi.org/10.1016/j.jss.2007.03.089

[4] T. Bruckhaus, N. H. Madhavii, I. Janssen, and J. Henshaw, "The impact of tools on software productivity," *Software, IEEE*, vol. 13, no. 5, pp. 29–38, Sep. 1996. [Online]. Last accessed June 2013. Available: http://dx.doi.org/10.1109/52.536456

[5] J. El-khoury, O. Redell, and M. Törngren, "A Tool Integration Platform for Multi-Disciplinary Development," in *31st EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 2005, pp. 442–450. [Online]. Last accessed June 2013. Available: http://dx.doi.org/10.1109/euromicro.2005.10

[6] M. Shaw and D. Garlan, *Software architecture*. Prentice Hall, 1996.

[7] iFEST Project: Industrial Framework for Embedded Systems Tools. [Online]. Last accessed June 2013. Available: http://www.artemis-ifest.eu

[8] CESAR Project: Cost-Efficient Methods and Processes for Safety Relevant Embedded Systems. [Online]. Last accessed June 2013. Available: http://www.cesarproject.eu

[9] S. L. Star and J. R. Griesemer, "Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39," *Social Studies of Science*, vol. 19, no. 3, pp. 387–420, 1989. [Online]. Last accessed June 2013. Available: http://dx.doi.org/10.2307/285080

[10] M. Biehl, J. El-Khoury, F. Loiret, and M. Törngren, "On the modeling and generation of service-oriented tool chains," *Journal of Software and Systems Modeling*, vol. 0275, Sep. 2012. [Online]. Last accessed June 2013. Available: http://dx.doi.org/10.1007/s10270-012-0275-7

[11] M. Biehl, J. El-Khoury, and M. Törngren, "High-Level Specification and Code Generation for Service-Oriented Tool Adapters," in *Proceedings of the International Conference on Computational Science (ICCSA2012)*, Jun. 2012.

[12] C. M. Pilato, B. Collins-Sussman, and B. W. Fitzpatrick, *Version Control with Subversion*, 1st ed. O'Reilly Media, Jun. 2004. [Online]. Last accessed June 2013. Available: http://www.worldcat.org/isbn/0596004486

[13] M. Biehl, "Semantic Anchoring of TIL," Royal Institute of Technology, Tech. Rep. ISRN/KTH/MMK/R-12/19-SE, Oct. 2012. [Online]. Last accessed June 2013. Available: http://www1.md.kth.se/~biehl/files/papers/semantics.pdf

[14] OMG, "Software & Systems Process Engineering Metamodel Specification (SPEM)," "OMG", Tech. Rep., Apr. 2008. [Online]. Last accessed June 2013. Available: http://www.omg.org/spec/SPEM/2.0

[15] AUTOSAR Consortium. (2011, Apr.) Automotive open software architecture (AUTOSAR) 3.2. [Online]. Last accessed June 2013. Available: http://autosar.org/

[16] P. Kruchten and P. Kruchten, *The Rational Unified Process*. Addison-Wesley Pub (Sd), Dec. 1998. [Online]. Last accessed June 2013. Available: http://www.worldcat.org/isbn/0201604590

[17] OMG, "Business Process Model And Notation (BPMN)," "OMG", Tech. Rep., Jan. 2011. [Online]. Last accessed June 2013. Available: http://www.omg.org/spec/BPMN/2.0/

[18] OASIS, "OASIS Web Services Business Process Execution Language (WSBPEL) TC," "OASIS", Tech. Rep., Apr. 2007.

[19] B. Curtis, M. I. Kellner, and J. Over, "Process modeling," *Commun. ACM*, vol. 35, no. 9, pp. 75–90, Sep. 1992. [Online]. Last accessed June 2013. Available: http://dx.doi.org/10.1145/130994.130998

[20] M. Biehl, W. Gu, and F. Loiret, "Model-based Service Discovery and Orchestration for OSLC Services in Tool Chains," in *International Conference on Web Engineering (ICWE2012)*, Jul. 2012.

[21] M. Biehl, J. Hong, and F. Loiret, "Automated Construction of Data Integration Solutions for Tool Chains," in *Seventh International Conference on Software Engineering Advances (ICSEA2012)*, Nov. 2012.

[22] D. Jackson and M. Rinard, "Software analysis: a roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 133–145. [Online]. Last accessed June 2013. Available: http://dx.doi.org/10.1145/336512.336545

[23] OMG, "Software & Systems Process Engineering Metamodel specification (SPEM) 2.0," OMG, Tech. Rep., 2008. [Online]. Last accessed June 2013. Available: http://www.omg.org/spec/SPEM/2.0/PDF

[24] C. J. Van Rijsbergen, *Information Retrieval*, 2nd ed. Butterworth-Heinemann. [Online]. Last accessed June 2013. Available: http://www.dcs.gla.ac.uk/Keith/Preface.html

[25] S. Voget, "AUTOSAR and the automotive tool chain," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '10. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2010, pp. 259–262. [Online]. Last accessed June 2013. Available: http://portal.acm.org/citation.cfm?id=1870988

[26] M. N. Wicks, "Tool Integration within Software Engineering Environments: An Annotated Bibliography," Heriot-Watt University, Tech. Rep., 2006. [Online]. Last accessed June 2013. Available: http://www.macs.hw.ac.uk:8080/techreps/docs/files/HW-MACS-TR-0041.pdf

[27] A. W. Brown and M. H. Penedo, "An annotated bibliography on integration in software engineering environments," *SIGSOFT Softw. Eng. Notes*, vol. 17, no. 3, pp. 47–55, 1992. [Online]. Last accessed June 2013. Available: http://dx.doi.org/10.1145/140938.140944

[28] B. Henderson-Sellers and C. Gonzalez-Perez, "A comparison of four process metamodels and the creation of a new generic standard," *Information and Software Technology*, vol. 47, no. 1, pp. 49–65, Jan. 2005. [Online]. Last accessed June 2013. Available: http://dx.doi.org/10.1016/j.infsof.2004.06.001

[29] R. Bendraou, J. M. Jezequel, M. P. Gervais, and X. Blanc, "A Comparison of Six UML-Based Languages for Software Process Modeling," *IEEE Trans. Softw. Eng.*, vol. 36, no. 5, pp. 662–675, Sep. 2010. [Online]. Last accessed June 2013. Available: http://dx.doi.org/10.1109/tse.2009.85

[30] L. Lucio, S. Mustafiz, J. Denil, B. Meyers, and H. Vangheluwe, "The Formalism Transformation Graph as a Guide to Model Driven Engineering," School of Computer Science, McGill University, Tech. Rep. SOCS-TR2012.1, Mar. 2012.

[31] A. Koudri and J. Champeau, "MODAL: A SPEM Extension to Improve Co-design Process Models," in *ICSP'10 Proceedings of the 2010 international conference on New modeling concepts for today's software processes*, ser. Lecture Notes in Computer Science, J. Münch, Y. Yang, and W. Schäfer, Eds., vol. 6195. Springer, 2010, pp. 248–259. [Online]. Last accessed June 2013. Available: http://dx.doi.org/10.1007/978-3-642-14347-2_22

[32] R. Bendraou, B. Combemale, X. Cregut, and M. P. Gervais, "Definition of an Executable SPEM 2.0," in *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*. IEEE, Dec. 2007, pp. 390–397. [Online]. Last accessed June 2013. Available: http://dx.doi.org/10.1109/aspec.2007.60

[33] B. Polgar, I. Rath, Z. Szatmari, A. Horvath, and I. Majzik, "Model-based Integration, Execution and Certification of Development Tool-chains," in *Workshop on model driven tool and process integration*, Jun. 2009.

[34] G. Karsai, A. Lang, and S. Neema, "Design patterns for open tool integration," *Software and Systems Modeling*, vol. 4, no. 2, pp. 157–170, May 2005. [Online]. Last accessed June 2013. Available: http://dx.doi.org/10.1007/s10270-004-0073-y

[35] A. Balogh, G. Bergmann, G. Csertán, L. Gönczy, Horváth, I. Majzik, A. Pataricza, B. Polgár, I. Ráth, D. Varró, and G. Varró, "Workflow-driven tool integration using model transformations," in *Graph transformations and model-driven engineering*, G. Engels, C. Lewerentz, W. Schäfer, A. Schürr, and B. Westfechtel, Eds. Springer-Verlag, 2010, ch. 10, pp. 224–248. [Online]. Last accessed June 2013. Available: http://portal.acm.org/citation.cfm?id=1985534