Heavy Meta

Model-Driven Domain-Specific Generation of Generative Domain-Specific Modeling Tools

Dissertation

zur Erlangung des Grades eines

D o k t o r s d e r I n g e n i e u r w i s s e n s c h a f t e n

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Stefan Naujokat

Dortmund

2017

Tag der mündlichen Prüfung:
20.03.2017

Dekan:
Prof. Dr.-Ing. Gernot A. Fink

Gutachter:
Prof. Dr. Bernhard Steffen
Dr. habil. Axel Legay
Prof. Dr. Jakob Rehof

Acknowledgements

First and foremost, I want to thank Bernhard Steffen for the supervision of my thesis. The many abstract discussions on ‘heavy meta’ matters regularly challenged the whole Cinco toolchain and DFS concepts, but always advanced the overall approach in the end. I also thank him for creating such a pleasant working atmosphere at the chair for programming systems in Dortmund. While interesting and innovative industrial projects easily distract one from finishing a thesis, I learned a lot aside from regular scientific business and enjoyed my time there.

Many thanks also go to the other referees of my thesis, Axel Legay and Jakob Rehof, as well as to Heinrich Müller for chairing my doctoral committee.

I am also particularly grateful to Tiziana Margaria for constantly pointing out opportunities for service orientation aside from classic usages, providing alternative perspectives on various topics, and finding ways to fund my initial post-Diplom research.

Also, I want to thank my longtime colleague Johannes Neubauer. Although (or maybe because) we usually start with quite opposite opinions on how to approach a problem – which regularly leads to extensive discussions – the collaboration in various projects always was very fruitful.

I especially owe thanks to Michael Lybecait and Dawid Kopetzki, who did most of the work for Cinco’s implementation. Without them, the whole Cinco framework would surely not have become what it is now.

Further gratitude goes towards the great office mates I had in the past years: Anna-Lena Lamprecht, Maik Merten, and Steve Boßelmann. All three of them literally rock. Beyond that, in particular the joint work with Anna-Lena in the PROPHETS and Bio-jETI projects, which already started with her co-supervision of my Diplom thesis, was fun and productive.

Finally, I like to thank my family: my wife Kerstin, who knows the mental absence and occasional grumpiness when writing a dissertation and thus was particularly patient and caring, as well as my parents, Gisela and Werner, for their extensive support throughout my educational career.

Abstract

Software is so prevalent in all areas of life that one could expect we have come up with more simple and intuitive ways for its creation by now. However, software development is still too complicated to easily and efficiently cope with individual demands, customizations, and changes. Model-based approaches promise improvements through a more comprehensible layer of abstraction, but they are rarely fully embraced in practice. They are perceived as being overly complex, imposing additional work, and lacking the flexibility required in the real world.

This thesis presents a novel approach to model-driven software engineering that focuses on **simplicity** through **highly specialized tools**. Domain experts are provided with development tools tailored to their individual needs, where they can easily specify the intent of the software using their known terms and concepts. This **domain specificity** (D) is a powerful mechanism to boil down the effort of defining a system to relevant aspects only. Many concepts are set upfront, which imposes a huge potential for automated generation.

However, the full potential of domain-specific models can only unfold, if they are used as primary artifacts of development. The presented approach thus combines domain specificity with **full generation** (F) to achieve an overall pushbutton generation that does not require any round-trip engineering. Furthermore, **service orientation** (S) introduces a ‘just use’ philosophy of including arbitrarily complex functionality without needing to know their implementation, which also restores flexibility potentially sacrificed by the domain focus. The unique combination of these three DFS properties facilitates a focused, efficient, and flexible simplicity-driven way of software development.

Key to the approach is a holistic solution that in particular also covers the simplicity-driven development of the required highly specialized DFS tools, as nothing would be gained if the costs of developing such tools outweighed the resulting benefits. This simplicity is achieved by applying the very same DFS concepts to the domain of tool development itself: DFS modeling tools are fully generated from models and services specialized to the (meta) domain of modeling tools.

The presented Cinco meta tooling suite is a first implementation of such a meta DFS tool. It focuses on the generation of graphical modeling tools for graph structures comprising of various types of nodes and edges. Cinco has been very successfully applied to numerous industrial and academic projects, and thus also serves as a proof of concept for the DFS approach itself.

The unique combination of the three DFS strategies and Cinco’s meta-level approach towards their realization in practice lay the foundation for a new paradigm of software development that is strongly focused on simplicity.

Attached Publications

Parts of this dissertation have already been published in cooperation with other scientists. They are listed here with according comments on my participation and will be referenced in a special way throughout the main body of this thesis to more easily distinguish them from other referenced literature.

I S. Naujokat, M. Lybecait, D. Kopetzki & B. Steffen. **CINCO: A Simplicity-Driven Approach to Full Generation of Domain-Specific Graphical Modeling Tools**. In. Int. J. on Software Tools for Technology Transfer (STTT), 2017.

Below cited as [AP I: NaLyKS2017].

The presented concepts were discussed among all authors. I was main author of all sections. The Cinco implementation has been primarily done by M. Lybecait and D. Kopetzki.

II B. Steffen & S. Naujokat. **Archimedean Points: The Essence for Mastering Change**. In: Transactions on Foundations for Mastering Change (FoMaC), volume 1, 2016.

Below cited as [AP II: SteNau2016].

The presented concepts were discussed among both authors. I co-authored all sections and was main author of sections 3, 4, and 5.

III S. Naujokat, J. Neubauer, T. Margaria & B. Steffen. **Meta-Level Reuse for Mastering Domain Specialization**. In: Proc. of the 7th Int. Symp. on Leveraging Applications of Formal Methods, Verification and Validation, Part II (ISoLA 2016), LNCS 9953, pp. 218-237, Springer, 2016.

Below cited as [AP III: NaNeMS2016].

The presented concepts were discussed among all authors. I co-authored all sections and was main author of section 4.

IV S. Naujokat, L.-M. Traonouez, M. Isberner, B. Steffen & A. Legay. **Domain-Specific Code Generator Modeling: A Case Study for Multi-faceted Concurrent Systems**. In: Proc. of the 6th Int. Symp. on Leveraging Applications of Formal Methods, Verification and Validation, Part I (ISoLA 2014), LNCS 8802, pp. 463-480, Springer, 2014.

Below cited as [AP IV: NaTISL2014].

The presented concepts were discussed among all authors. I co-authored all sections and was main author of sections 2, 3, 4.1, and 5.

V S. Naujokat, J. Neubauer, A.-L. Lamprecht, B. Steffen, S. Jörges & T. Margaria. **Simplicity-First Model-Based Plug-In Development**. In: Software: Practice and Experience. John Wiley & Sons. Dec 2013.

Below cited as [AP V: NNLSJM2013].

The paper is a thoroughly extended and reworked version of [NLS+12b]. Its concepts were discussed among all authors. Most sections were co-authored by J. Neubauer and myself. I was main author of sections 3.1, 4.2 and 4.3.

VI S. Naujokat, A.-L. Lamprecht & B. Steffen. **Tailoring Process Synthesis to Domain Characteristics**. In: Proc. of the 16th IEEE Int. Conf. on Engineering of Complex Computer Systems (ICECCS 2011), pp. 167-175, IEEE, 2011.

Below cited as [AP VI: NaLaSt2011].

The presented concepts were discussed among all authors. I co-authored all sections and was main author of sections 2 and 3.

Contents

1 Introduction

Increasingly huge systems with millions of lines of code create the need for more efficient approaches to the development, management and maintenance of software. Besides ‘classic’ programming solutions – like modularity, design patterns, tier architectures, etc. – various forms of model-based software engineering approaches emerged promising improvements through a more comprehensible layer of abstraction. The role of models in these approaches ranges from standardized ways of visualization and documentation over partial code generation to first class citizens in a fully generative setting.

UML [RJB04] is probably the most widely known family of modeling languages for software engineering. However, studies [DP06, FL08, Pet13] indicate that it has not been widely adopted in the industry. They reflect the opinion that the benefit over classic development approaches (if present at all) is not worth the effort and that known standards are overly complex, impose additional work, and lack the flexibility required in ‘real’ projects. [Pet13] quotes a participant from an industry survey saying that UML “Doesn’t add anything except religion”.

This thesis combines three strategies that counteract these problems and (together) have the potential to increase the acceptance of model-based approaches by practitioners:

Domain Specificity removes overhead complexity of the development approach and its modeling languages, which effectively simplifies expressing the actual purpose of the developed system. This is done by focusing only on problems of the respective domain, as a more targeted formalism becomes easier to handle and in particular enhances the communication between domain experts and technical experts. In general, domain specificity introduces restrictions that sacrifice generality for simplicity.

Full Generation removes the huge round-trip effort imposed by many model-based approaches, in which models often just serve as a first step in the design phase of a system. This round-trip adds lots of unnecessary workload and is a common source of inconsistencies and errors. Full generation, on the other hand, postulates a concept in analogy to compilers for classic programming languages: models need to be the primary artifacts of development, i.e., they must be expressive enough, so that within their respective domains a running system can be derived fully automatically from only those models.

Service Orientation removes the necessity to make a modeling language more complex to cover (rare) cases, for which writing code in a programming language might be suitable, or an adequate third-party realization already exists. The service-oriented inclusion of such solutions eliminates the need to understand how they are implemented. This way, the flexibility often required in realistic scenarios can be achieved without the loss of simplicity.

In the following, the term **DFS tool** is used for a domain-specific, fully generative, and service-oriented modeling tool realized according to those factors. The classification should neither be regarded as absolute nor as complete, but it identifies good primary factors for the power of modeling approaches. While the development of a system can truly benefit from all three aspects being sufficiently supported and easily accessible, common frameworks for model-based development rarely support more than one.

In particular the aspect of domain specificity implies that DFS tools are highly specialized. It is thus crucial that their development efforts do not exceed the saved efforts when using them (summed up over all applied cases/problems). However, state-of-the-art approaches for modeling tool development, which are usually based on metamodeling, are often still considered to be too complex and costly to use.

To simplify the development of DFS tools I present an approach that **applies the DFS factors to the domain of tool development itself**. This essentially raises those three factors to the meta level and extends the ideas of metamodeling to a generalized notion of meta-level specifications, which

1. are domain-specific by restricting on a subset of all possible model types (**D**),
2. are fully generatable into the modeling tool (**F**), and
3. integrate existing metamodeling-based solutions in a service-oriented fashion (**S**).

This leveraging to meta level provides, in contrast to existing methods, a very pragmatic approach to the development of domain-specific modeling tools. Its goal is to pay off even when utilized for a high degree of specialization, i.e., developing highly optimized modeling solutions for very specific domains.

The **Cinco meta tooling suite** provides a first implementation for this approach to meta domain specialization. It focuses on the development of graphical modeling tools comprising different types of nodes and edges. Cinco has been applied very successfully in various industrial and academic projects, and thus serves itself as a proof of concept for the DFS factors.

The remainder of this thesis is structured as follows: Sects. 1.1 and 1.2 of this chapter summarize my contribution and set the context for the attached publications. Individual background, motivation, and discussion on each of the DFS aspects is presented in Chap. 2. Chap. 3 then first sketches from a bird's eye view Cinco's concept of meta domain specialization, before elaborating individually on each of the meta-level DFS realizations. In Chap. 4 Cinco's features specifically supporting the development of DFS tools are presented. After a brief discussion of related work in Chap. 5, perspectives for enhancements of Cinco and future research are presented in Chap. 6. The thesis concludes with a summary in Chap. 7.

1.1 My Contribution

In essence, the thesis makes three contributions towards a novel model-driven software engineering approach that focuses on simplicity through highly specialized development tools:

- DFS tools combining domain specificity with full generation and service orientation allow for focused simplicity-driven development (presented in Chap. 2).
- Development of according modeling tools is facilitated by self-application of the DFS factors: the tools are generated with a specialized meta DFS tool (presented in Chap. 3).
- Additional dedicated support can be provided on the meta level to further aid in the creation of model-based development tools following the DFS factors (presented in Chap. 4).

Providing a holistic solution that in particular covers the development of the required highly specialized tools makes the approach applicable in practice. Important in this context is the Cinco meta tooling suite. It provides an initial realization of the DFS factors on the meta level and is specifically designed to support the development of DFS tools.

The ideas on how to improve the development of domain-specific modeling tools, which led to the CINCO project, arose from one particular modeling tool I developed for an industry project in 2011–2013. While CINCO profited from many discussions with colleagues, the fundamental concept of fully generating a modeling tool from simplified (restricted, domain-specific) specifications was developed by me. Since then, the CINCO project significantly drove the research and teaching of the 'programming systems' group in Dortmund.

The initial implementation has been primarily done by Michael Lybecait and Dawid Kopetzki. It was partly based on their Diplom and MSc theses [Lyb12, Kop14], which I both supervised. Many other Diplom, BSc, and MSc theses [Tam14, Wir15, Zwe15, Wor15, Wec16], as well as student project groups [BDG+15, STK+16, WMN16] were based on CINCO and conducted under my supervision.

1.2 Context of Attached Publications

This thesis is part of a cumulative dissertation that comprises already published articles written in cooperation with other scientists. While page vii lists these publications with comments on my participation (as demanded by TU Dortmund's doctoral degree regulations), this section explains the overall context of these publications and my thesis.

[AP I: NaLyKS2017] is the main appended publication that introduces the ideas and realization of the CINCO meta tooling suite. It in particular comprises a detailed description of CINCO's specification formalisms that are used for the full generation of graph-based modeling tools, a comprehensive presentation of various academic and industrial applications, as well as an extensive comparison to other metamodeling approaches.

[AP II: SteNau2016] formalizes the notion of domain specificity in the context of modeling language evolution. 'Archimedean Points' (APs) are introduced as a (meta-level) concept for things that do not change, i.e., can be relied upon during evolution. A modeling tool developed with CINCO serves as a running example. During its evolution from a tool for simple place/transition nets to one for the modeling of BPMN processes, several APs are discussed, maintained, and – when required – even adapted.

[AP III: NaNeMS2016] provides a general discussion on programming and modeling by distinguishing *how* something is realized from *what* its purpose is. Based on this, future enhancements of CINCO are envisaged that provide a unification of programming and modeling via a meta-level conceptual core. These enhancements comprise extending CINCO in a bootstrapping fashion, on the one hand by generalizing concepts that were initially developed for single projects done with CINCO, on the other hand by using CINCO to generate dedicated specialized modeling languages for CINCO itself.

[AP IV: NaTISL2014] presents one of the first case studies done with CINCO. Custom graphical interfaces were developed for timed automata (TA), probabilistic timed automata (PTA), Markov decision processes (MDP) and simple labeled transition systems (LTS). Based on a (also during this study developed) Parallel Systems Modeling (PSM) superset language, various generators produce code for the verification frameworks UPPAAL, SPIN, PLASMA-LAB, and PRISM. The multi-step generative concept of this project laid the foundation for the notion of meta$_n$modeling, which will be subject to future research. The idea is briefly sketched in Sect. 6.4.

[AP V: NNLSJM2013] utilizes the generative features of the PROPHETS synthesis to modify the jABC process modeling framework. It presents an automized approach to the creation of plug-in functionality to more easily achieve domain specialization with jABC. This can be regarded as a conceptual precursor to full tool generation, as done with Cinco.

[AP VI: NaLaSt2011] presents a first step into the direction of modifying (parts of) a plug-in for domain-specific concerns. The jABC PROPHETS plug-in provides loose programming for jABC process models. However, we have shown here that the plug-in itself should be specialized towards the targeted domain. This case study basically represents a first manually realized incarnation of the meta plug-ins' generative approach to specializing plug-ins for individual domains.

2

DFS: Conceptual Background and Discussion

This chapter discusses the conceptual foundations of the three factors which I consider essential for tools to facilitate simplicity-driven model-based software development: domain specificity, full generation, and service orientation. In each section, the respective concept is introduced according to existing approaches and notions. As there is often no generally agreed upon definition, I pinpoint the aspects that I consider most important in the context of my thesis. Furthermore, I discuss why each of the DFS factors is key to achieving the overall goal of simplicity in model-based software engineering and provide classifications of existing modeling approaches, i.e., whether, and to what extent, they support the respective concept.

2.1 Domain Specificity

Restricting a formalism to problems of a certain domain has essentially two effects:

- The formalism is easier to understand by domain experts, as the purpose of a system is expressed with terms and notions from their area of expertise, and overhead complexity (induced by the formalism) is generally reduced.
- Many aspects are set upfront, so that a huge amount of functionality can be automatically generated from simple and small specifications.

This section first motivates this notion of domain specificity relying on widespread concepts of model-driven software engineering and domain-specific languages. Then, the challenge of finding the adequate level of domain specificity is discussed. Furthermore, an assessment of existing approaches is provided regarding their support for this kind of specialization.

2.1.1 MDSD and DSLs

The aspect of domain specificity is commonly associated with (textual) *domain-specific languages* (DSLs) [FP11]. However, DSLs and *model-driven software development* (MDSD) have many overlapping ideas and can not be cleanly separated. On the one hand, domain specificity can be regarded as an orthogonal aspect to model-driven development, as the degree of domain specificity and the degree of modeling can be seen somewhat independently. While the former counters ’general-purposeness’, the latter influences the ‘amount of programming’.

On the other hand there are shared central concepts, such as abstraction and code generation, and the overall goal to specify and modify a system's behavior in a more comprehensible way. Models and DSLs both tend to move this specification from a (technical) description of *how* it is done to a requirements specification of *what* the intended behavior is. We discussed in [AP III: NaNeMS2016] that this distinction is a matter of perspective: "what is a model (a what) for the one, may well be a program (a how) for the other". One of the first who brought together domain specificity and modeling were Kelly and Tolvanen with the MetaEdit framework [KLR96] and its underlying concept of *domain-specific modeling* (DSM) [KT08].

Common to modeling and DSLs is that the system under development is (partly) defined through an abstract representation, either as models or by specialized text formats. However, even for the textual DSLs, Fowler [FP11] suggests to use a "semantic model" to represent the parsed language in memory. This model essentially corresponds to the abstract syntax in a metamodeling context. The opposite direction applies as well: in model-based approaches to DSL development (such as with Eclipse EMF [SBPM08]) one usually can define multiple concrete syntaxes for a given abstract syntax metamodel, i.e., multiple representations for the same modeling language. Graphical syntaxes can lead to something like UML or BPMN [36], but textual syntaxes are essentially the DSLs considered by Fowler. While this thesis focuses on specialized graphical languages, many of the presented concepts can similarly be applied to textual languages.

Abstract representations are – in particular if they hide technical detail – also a means of communication. Information technology, automation, etc. have become more and more prevalent in nearly all fields of life. Often, the corresponding technical system needs to be customized to individual demands, but there is a huge semantic gap between the people knowing these demands (i.e., the domain experts) and the people who realize it (i.e., technical experts). Bridging this gap is one of the most pressing challenges of software development, for which domain-specific languages seem to be a promising approach [FP11].

2.1.2 Degree of Domain Specificity

The question whether an approach should be considered as domain-specific or general-purpose is often a matter of perspective. Most languages commonly denoted as general-purpose can as well be regarded as domain-specific, as they always abstract from certain detail that is then included by a generator or compiler. Therefore, any form of model-driven development – and even general-purpose programming – can be regarded as domain-specific. It just depends on how broad one defines a domain.

However, this discussion does not provide any guidance for the requirement postulated by this thesis to include domain specificity into model-driven engineering for it to be more applicable in practice. For this, most existing modeling approaches are too generic. In a domain-specific setting, one needs to concentrate on core concepts instead of making the scope of a modeling language considerably broader just to be able to cover a few rare corner cases. The language will then be unnecessarily complex for the bulk of regular cases[1].

Therefore, finding an adequate level of domain specificity always needs to be part of a project. A general observation is that the higher the degree of domain specialization, the bigger is the generative lever: more code can be generated from simpler specifications, making the solving of bigger problems within the targeted domain manageable. In return, problems outside this domain are usually not covered anymore. This introduces a trade-off: While the first proposition implies that the tooling should be very highly specialized, the second prohibits extreme specialization in practice. By introducing more efficient ways of tool development (as done with CINCO, cf. Sect. 3 and 4), this trade-off gets less severe.

[1]It might even be that a dedicated language for such corner cases would on its own be simple again.

2.1.3 Internal and External DSLs

Fowler [FP11] classifies DSLs into two categories: *internal* and *external*. The former are solely based on some general-purpose host language. There, DSLs are either special forms of APIs (e.g. realized as *fluent interfaces* [28]) or the language directly supports building such DSLs (e.g. Lisp [Wei67], Ruby [15], Swift [17]). External DSLs, on the other hand, always involve a different syntax that needs to be processed (i.e., parsed) and then included or generated into some general-purpose host language.

We discussed in [AP III: NaNeMS2016] that for the goal of reducing complexity (i.e., increasing simplicity), external DSLs should generally be favored, as internal DSLs still require too much understanding of the surrounding host language. Therefore, this thesis primarily focuses on external domain-specific (modeling) languages. However, service orientation can restore lost flexibility by providing a notion of internal DSLs for an external DSL host language (cf. Sect. 2.3).

In general, the fact that language development is complicated (and thus expensive) is largely independent[2] of the degree of domain specialization; this in particular holds for graphical languages, for which the development technologies often have a high entry hurdle. Thus, more general-purpose solutions are usually developed. Generative approaches based on metamodeling aim at reducing such costs, but existing solutions so far have simply not gone far enough. In classic metamodeling approaches, e.g., as defined with MOF [2] by the OMG or realized with EMF in the context of Eclipse, the metamodel does only specify the abstract syntax of the model. While lots of code can already be generated from that, there is still a big gap to be filled for reaching a sophisticated modeling tool.

Also, modern *integrated development environments* (IDEs) introduced a certain degree of convenience to the development process[3]. Assistance and productivity features that are by now common for classic software development (such as live validation, refactoring, project management, versioning, etc.) should also be available in model-driven software engineering contexts. Fowler introduces the notion of *language workbench* [26, 29] for frameworks that support the development of external DSL tools with IDE-like features. Although he generally targets more towards textual DSLs, the Cinco framework presented in this thesis can be regarded as a language workbench for external graphical domain-specific modeling tools.

2.1.4 Classification of Existing Approaches

Classifying the ability to influence the degree of specialization is particularly easy for all meta-level approaches compared in [AP I: NaLyKS2017] and Sect. 5: providing means to develop external domain-specific languages is a core concept for all of them.

For modeling approaches not providing metamodeling facilities to define a domain model, this classification is not as obvious, though.

In the context of OMG languages, UML can be regarded as providing internal DSL mechanisms through the use of *profiles*. Their family of business management languages (BPMN, CMMN [38] and DMN [40]), on the other hand, so far does not define means of domain specialization, so that individual companies have come up with own adaptations (see, e.g., BPMN element templates by Camunda [43]). For the standards, in turn, the existence of a profile specializing UML to the 'domain' of BPMN business processes [34] shows that OMG considers BPMN already domain-specific, while in the context of this thesis it would be considered rather general-purpose.

[2]This might not apply for extremely simple languages, like a dedicated small configuration file for a single purpose. However, this is a case where the degree of specification is too high, so that one can not build a system only on this kind of languages.

[3]Fowler [27] speaks of "post-IntelliJ era" honoring the strong supporting features already early versions of JetBrains' IntelliJ IDEA [30] introduced for developers.

MetaFrame [SMCB96, SM99] and the *Java application building center* (jABC) [MSR06, SMN+07, KJMS09] support domain specificity through dedicated plug-ins, service taxonomies, and component libraries. In particular the latter is essentially a method of providing internal DSLs that strongly corresponds to the tools' support for service orientation (cf. also Sect. 2.3). I also contributed work regarding (automized) adaptation of jABC to domain-specific needs, in particular through specializing the PROPHETS plug-in [LNMS10, NLS12a] and through synthesis of plug-in functionality. This was presented in [AP VI: NaLaSt2011] and [AP V: NNLSJM2013], respectively.

2.2 Full Generation

Many modeling approaches impose a huge round-trip effort to software development, which adds unnecessary workload and is a common source of inconsistencies and errors. The proposed concept of full generation, on the other hand, postulates that models need to be made the primary artifacts of development. This implies that they are expressive enough, so that a running system can be derived fully automatically from only those models. This approach does not only include generation of code, but actually everything that is either directly executable or can in succeeding steps be automatically transformed to being executable.

This section first discusses the problem of partial generation and its induced round-trip, before presenting common approaches to full code generation. A unified view is then presented that motivates the use of the term 'full generation' without the inner 'code'. Again, the section closes with a classification of existing approaches.

2.2.1 Model-Driven Approaches

Many terms are widely used for software engineering approaches that involve models. However, again, there is no clear and universally accepted distinction, so I will point out some distinctions that seem most widely used.

In model-*driven* approaches models have a formal role, as they are considered key artifacts of the development that serve as the basis for automated processing, such as code generation, transformation, validation, etc. Here, so many terms for subtypes have emerged that Völter [44] introduced MD* "as a common moniker for MDD, MDSD, MDE, MDA, MIC, LOP and all the other abbreviations for basically the same approach". However, the term seems not widely established. So, while I generally agree with the identified commonalities, I will just refer to model-driven (software) development/engineering.

Sometimes, the word model-*based* is used for scenarios with informal roles of models, e.g., when they are used primarily for documentation. Cabot [BCW12] [25] even regards model-based approaches as a superset of model-driven approaches. Generally, this thesis clearly contributes to model-driven approaches. However, I will use the term 'model-based' whenever a weaker classification is appropriate.

2.2.2 Partial Code Generation

One of the earliest forms of model-driven software development involves the generation of code *skeletons* (or *stubs*, as they are sometimes also called) that subsequently need to be fleshed out manually. Here, the generation is only a one-time step in development that can not be applied in later phases of a project, as regenerating code from models basically results in loss of all work done since last generation.

Various techniques have been developed that aim at overcoming this problem. One approach is the use of *protected areas* [BCW12] (also called *protected regions* [KT08]): Generated parts of the code are marked (e.g. by dedicated comments), so that on regeneration the code generator can identify them and replace only the contents of such regions. Manual changes are solely allowed outside those areas, but this is often not technically enforced, so that it depends on the programmers' discipline to obey this rule.

A similar approach is the *generation gap pattern* [Vli98], which also aims at keeping manually written and generated code apart, but more cleanly separates them by using dedicated language entities. In an object-oriented context, it proposes to extend generated classes with hand-written ones that add manually implemented functionality. An adaptation documented by Fowler [FP11] additionally uses a hand-written superclass for the generated class to also separate preset static code parts (i.e., ones that are independent of code generation) from code that is actually influenced during generation.

2.2.3 Full Code Generation

While already the separating approaches are clearly superior to one-time skeleton generation, they still require manual round-trip effort when structural changes are made that affect the existence of the generation gaps and protected areas. Also, they are error-prone (e.g., regarding reliable automatic detection of protected areas) and depend on user discipline.

An approach that overcomes these problems is full code generation. The underlying idea is to specify everything consistently on the model level and automatically generate the application from it. This in particular means that the output of the code generator is fully functional and does not need to be manually modified. Subsequent changes are only made on the model level, followed by a regeneration step. By explicitly forbidding that generated code may be changed, round-tripping is naturally circumvented.

Full code generation makes the models 'first class citizen', i.e., they become the primary artifacts of the development process. This should consequently also result in treating them exactly like source code in classic programming projects. For instance, only the models – not the code generated from them – should be versioned with a project's SCM repository. While it is, for instance, common practice not to version compiled `.class` files in a Java project, experience shows that the corresponding concept in model-driven development is not obvious for most developers. Their perception seems to be different in this context, as models appear to be 'farther away' from source code. However, in fully generative approaches, their role is exactly the same.

2.2.4 Unified View

So far I discussed mainly the (full) generation of code – i.e., producing text in the syntax of some general-purpose programming language – from models. However, this is not the only scenario where full generation applies. Basically, every automized transformation from a source language to a target language can be considered full generation, if the output is executable on its own. In [JLM+16] we discussed that even techniques for program/process synthesis from formal specifications can be regarded as fully generative in this context.

The generation may as well comprise intermediate languages with different generators stepping through the process. In fact, generating source code always additionally requires another generator – the language's compiler – to actually produce executable (machine) code. Above that, in a virtual machine-based language like Java, also the interpreter or just-in-time compiler of the JVM is required for the bytecode produced by the Java compiler to be executed.

Under this unified view of full generation, even (code) generation and (model) transformation is essentially the same; just with differences in source and target representations as well as applied techniques (e.g., visitor pattern, templates, etc. [LKR14]).

Full generation also can involve multiple source formats as well as multiple target formats (cf. [AP III: NaNeMS2016]), as long as the sum of target artifacts (be it as models, as programming language classes, DSL texts, etc.) is either directly executable or in succeeding steps can be made executable fully automatically. This also includes that parts of the 'source side' models are actually source files of a general-purpose programming language, which may be interconnected with further, more abstract models. As discussed in Sect. 2.1, source code can be considered as a DSL as well; just with lesser degree of specialization. Thus, in essence, the role of a full generator for a modeling environment is the same as the role of a compiler for a programming language. An interesting (but disturbing) observation is that the credo 'never change generated code' is usually not questioned for a compiler, but regularly violated in a model-driven context.

Compilers and accompanying build tools like make, ant, or maven (which together facilitate full generation for whole projects, and in particular enable continuous integration [Fow00]) have proven indispensable for efficiently working on software projects in practice. Therefore, I consider an according equivalent as a key factor for model-driven software engineering. Although most model-driven solutions following the DFS factors will probably generate code, the concept itself is not restricted to code. I will thus simply use the more general term 'full generation'.

Of course, full generation does not mean that every piece of executed code is actually generated. There will almost always be frameworks and libraries used in the target languages, which can, but not necessarily have to be, generated themselves. In the context of Java, for instance, this means that during compilation the generated bytecode is linked to existing `.class` files which either have been compiled from `.java` sources themselves or have been generated with a different one from the wealth of JVM languages, such as Scala [19], Groovy [18], Kotlin [12], or even the Jasmin assembler [11]. Actually, the inclusion of arbitrary target space (native) artifacts – without caring about their realization – is one of the ideas behind service orientation (cf. Sect. 2.3)

2.2.5 Classification of Existing Approaches

When questioned about models in software engineering, most people will come up with UML as 'de facto standard'. However, UML's initial goals were primarily targeted towards providing a standard for software visualization and documentation [35] and many UML tools provided only very basic code generation, e.g. for class diagrams, which often did not even support protected regions or generation gaps.

More recently, there has been a change of focus on the part of OMG (who coordinates the development of the UML standard). Adaptations like the UML *action language* (ALF) [37] and *foundational UML* (fUML) [41] are more going into the direction of having a well-defined execution semantics (for a subset of UML) that allows models to be automatically transformed into executable code. An earlier (not OMG-standardized) approach was *Executable UML* [MB02]. However, these approaches still have not been widely adopted in practice.

Umple [FBL10, FBLS12] is a family of textual modeling languages aiming at seamlessly integrating modeling and programming. Its modeling parts also focus on well-defined and fully generatable aspects of UML. As it is designed to enhance programming with modeling, full generation is indeed part of the core concepts.

jABC [MSR06, SMN+07, KJMS09] with the Genesys framework [JMS08, Jö13] has always targeted full code generation. It is actually one of the basic principles[4] for the *one-thing approach* (OTA) [MS09] that proposes to build and refine a single model structure consistent with all requirements and constraints, and its automatic generation into the complete running product.

[4]Another one is service orientation, see Sect. 2.3.

Of the approaches compared in Sect. 5 and [AP I: NaLyKS2017], *domain-specific modeling* (DSM) [KT08] is the only one that very decidedly promotes full code generation.

2.3 Service Orientation

The ability to just include ready-made services removes the necessity to make models (and the according languages) more complex to cover cases, for which writing *native* code in a programming language might be suitable, or an adequate *third-party* realization already exists. This essentially spans two dimensions which abstract from *how* a service is realized, and *who* provides it (cf. Fig. 2.1). It even covers *reuse* of self-developed models simply as services. Overall, this form of service-oriented thinking facilitates a degree of flexibility often required in realistic scenarios and can be achieved without the loss of simplicity.

This approach is heavily influenced by jABC's underlying concept of *lightweight process coordination* (LPC) [MS04], but does not entirely match with other widespread notions of the term 'service'. Thus, this section first sets the focus in relation to those other notions and identifies common properties of services. Then, libraries are introduced as a means of facilitating third-party services, as well as native constructs as a means for the inclusion of target space artifacts. Afterwards, existing approaches are analyzed regarding these two dimensions.

2.3.1 Setting the Focus

The term 'service', which already has several meanings in non-technical contexts [1], has been widely used in various areas of information and communication technology, e.g., for *intelligent network services* [SM99, MSR05], *service-oriented computing* (SOC) [HS05], *service-oriented architectures* (SOA) [Erl07], *web services* [24] (including the hype of *Web 3.0* with *semantic web services* [BHL01] in the early 2000s), and *REST services* [Fie00]. While SOA is probably the most widespread, it can be disputed whether it was usually applied successfully in practice or should be considered 'dead' by now [31, 33, 32]. Nonetheless, services in general are a concept of ever-growing importance. The emergence of *cloud computing* [AFG^{+}09] and *everything as a service* (XaaS) – with its variants *software as a service* (SaaS), *platform as a service* (PaaS), and *infrastructure as a service* (IaaS) [RCL09] – have boosted the perception of services to something widely accessible for a broad audience.

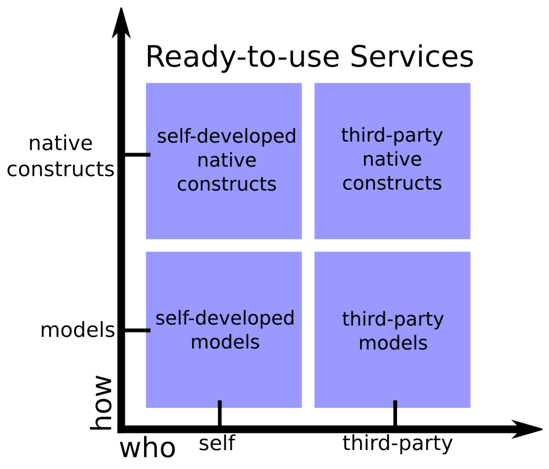

Fig. 2.1: Services are 'just used' the same way, independently of who developed them how.

All those notions have one particular thing in common: they strongly focus on availability of services via network infrastructures, which is the internet in most cases, but can also refer to local networks or other networking technologies. However, as one of my three key factors for model-driven software engineering I consider a more generalized notion of *service-oriented thinking* that is independent of online accessibility and focuses on the aspect of 'just using a service'. This in particular allows models to be treated like services and follows the idea of *service independent building blocks* (SIBs) of the *lightweight process coordination* (LPC) approach [MS04].

2.3.2 Properties of Services

While being available via network is not explicitly excluded with the SIB notion of services, it is no core feature. However, many properties and requirements [HS05, Erl07] important for online services can easily be transfered to this more general view:

Black Box is one of the most central features of services. It means that a service can be utilized without knowing its internals, i.e., how exactly it is realized. In some cases one can look into this realization, e.g., because it is defined by some other team member in the same project. However, it is never actually required for the service user. This concept is the foundation of achieving simplicity through services, as arbitrarily complex realizations can be hidden from the user.

Interface defines the technical, syntactical way of interacting with a service. This mainly comprises how and where to access the service, as well as how and which kind of data to pass.

Service Level Agreement (SLA) is a term adopted from (potentially) non-technical areas, where it denotes a contract between service provider and client that defines the nature of a service. In the context of services considered in this thesis it describes what the service does and basically serves as the semantic counterpart to the interface definition. The SLA can be provided in a formal way, e.g., with pre and post conditions, but is more commonly just plaintext documentation.

Atomicity states that a service provides functionality on its own and does not rely on others. However, this does not mean that the service is not allowed to utilize other services (see below), but if it does, this is – as part of the black box idea – not exposed to the service user.

Autonomy serves as the semantic equivalent to atomicity. Basically, it means that there is no feature interaction with other services, or, more generally, the service is free of side effects. Sometimes, this also includes the property of statelessness. However, autonomy is usually very difficult to achieve, and thus no strict requirement. Therefore, it is all the more important to properly document the service's effects.

Composition means that services can themselves be composed of other services, which is the main premise for hierarchical service design [SMBK97]. It also comprises reusability of services and is a natural result of independent atomic services. Composing arbitrary services again to other services, which is commonly called *service orchestration*, greatly contributes to the flexibility of the overall approach.

Other properties not considered here cover, for instance, the automatic *discovery* of services, which is a core feature of online approaches, e.g., for SOA and the Semantic Web. However, the definition above naturally includes the whole discovery mechanism to be itself provided as a service. In particular discovery can, if required, be easily realized with higher-order approaches [NSM13, NS13a], which facilitate passing around processes (and thus services) just like data.

2.3.3 Service Libraries

While online discovery and remote access to services is not primarily considered here, the context of model-driven software engineering does indeed require a corresponding mechanism to share and distribute services, so that reuse and separation of concerns are easily facilitated. Basically, this need for inclusion of third-party services is covered by libraries, i.e., thematically packaged bundles of ready-to-use service components, from which developers can manually choose required services and parameterize them to their individual needs. This library approach might again as well include mechanisms for automatic distribution of libraries, so that the advantages of online discovery also can be applied on this level.

In any context resembling regular programming (including programming in a general-purpose language itself) libraries are a common concept. They do, however, not necessarily fulfill all service requirements. In particular atomicity often does not exist, when functional and structural concerns are intertwined and require complex combinations of usage.

2.3.4 Native Constructs

Programming languages usually have some form of native integration that allows developers to directly provide implementations in the formalism of the target space. For example, C compilers can include assembler code that was inlined into regular C code, and Java allows calls to system-dependent functionality via the *Java native interface* (JNI).

To maintain the flexibility required in practice without giving up on the claim for full generation (cf. Sect. 2.2), services in a model-based context require an equivalent for these native constructs. Here, this is even more important than for the mentioned programming languages – where the use of native calls is limited to very specific scenarios and generally rather discouraged – because modeling languages are by design more abstract, i.e., not as expressive as programming languages[5].

Generally, one does not want to include into a modeling language the ability to specify every intricate detail, as this severely hampers the aspect of simplicity. For instance, one would not want to express complex and highly optimized algorithms purely with models. However, using such algorithms as components within globally orchestrating models abstracts away the intricate details and purely focuses on the functionality.

Libraries of hierarchical models, which may even make use of native constructs, can be regarded as a means of adding internal DSLs to an external DSL approach. So, changing the modeling tool itself, which in particular comprises adapting its syntax and the code generator, is not required for maintaining and extending modeling functionality.

Overall, hierarchical services and native components can balance the trade-off between flexibility and simplicity. In combination with the separation of concerns induced by the service-oriented thinking, they are a powerful means for flexibility that does not per se introduce additional complexity for the modeling level.

2.3.5 Classification of Existing Approaches

As already discussed, in nearly every programming language service orientation can be achieved through libraries forming internal DSLs. Also, if required, native constructs provide the flexibility to achieve

[5]Fowler [FP11] reports Turing completeness as a common distinguishing factor between domain-specific and general-purpose approaches. But even a Turing complete modeling language might not be well-suited for certain kinds of tasks.

high specialization to the underlying architecture. In modeling approaches that are conceptually close to programming, such as Umple [FBL10, FBLS12], both are similarly supported.

On the side of more abstract modeling languages, however, actually very few approaches actively promote service-oriented thinking, though some come with a library mechanism that probably could be utilized for that, as the concept itself is more a matter of mindset than technology. The fallback to native code does exist in some approaches, but is often realized by some form of 'script activities', whereas properly supported ready-to-use services in the form of 'business activities' are poorly supported at best [DS12].

A modeling technology with a long history of strict service-oriented thinking is the family of MetaFrame [SMCB96, SM99] and jABC [MSR06, SMN$^+$07, KJMS09] tools. While the former was originally developed to model intelligent network services for the telecommunications sector, the latter is the original implementation for the *one-thing approach* (OTA) [MS09], which can be considered a special discipline of model-driven software engineering that combines the integration of arbitrary services with full code generation [JMS08, Jö13]. Both are built upon this concept of libraries of SIBs, which provide access to services and can be graphically composed to *service logic graphs* (SLG). The SIB library concept provides an intuitive notion of service orientation with SLGs serving as a high-level coordination and orchestration language. The full code generator then primarily needs to produce the control structure between SIB executions. As a SIB is either a process model (forming hierarchical services), or connected to an implementation of the target platform, which forms the native constructs in this case, both dimensions of service orientation introduced above are supported.

In the context of newer jABC versions, *second-order servification* [NS13b] makes use of *higher-order process engineering* [NS13a, NSM13] to facilitate a very flexible way of handling services (in the form of whole processes) at runtime by instantiating and passing them around just like data.

Furthermore, the *electronic tool integration* (ETI) platform [SMB97] bridges the gap between accessibility via network and jABC's more general notion of services. The ETI system was one of the first approaches towards a simplified access to complicated tools, namely from the formal verification world, to make them commodity. While SIBs in general are executed locally, the jETI framework [MNS05, KMSN07] provides a simple form of using online services in SLGs, which in particular has been utilized to include bioinformatics services in scientific workflows created with jABC [MKS08, Lam13].

3 Cinco: DFS on the Meta Level

The previous chapter discussed that a model-driven approach following the three DFS factors can significantly improve software development projects by reducing complexity and overhead work, while at the same time being efficient and flexible. However, this approach requires (potentially very highly) specialized modeling tools, and nothing would be gained if the costs of developing such tools outweighed the resulting benefits, which is usually the case when using current state-of-the-art metamodeling solutions.

By applying the very same DFS concepts to the domain of modeling tool development itself, however, the approach is made feasible. In essence, a corresponding 'meta-level' DFS tool needs to be domain-specific by specializing on certain kinds of modeling tools, fully generate the modeling tool from specification models, and integrate existing solutions in a service-oriented way.

Cinco is a first proof of concept implementation for such a tool. While it in particular supports the development of modeling tools satisfying the DFS properties (see Chap. 4), being itself designed as a DFS tool has two immediate effects:

- Cinco's simplicity-oriented nature allows for 'early wins'. Initial versions of specialized modeling tools can easily be developed and applied in a project right from the beginning and agilely adapted to changes in the course of the project.
- Cinco is itself a proof of concept for the simplicity and the big generative lever induced by the DFS properties. No other meta tools – including particularly widespread metamodeling solutions, such as EMF [SBPM08] or DSM [KT08] – sufficiently support all three factors, which makes them more complicated, less efficient to use, and thus more expensive to apply to a project.

In the following, Cinco's meta DFS approach is presented from a bird's eye view in Sect. 3.1, before the subsequent Sects. 3.2–3.4 individually detail on how each of the three properties is realized on the meta level within Cinco.

3.1 Meta DFS from a Bird's Eye View

One of the most important design principles of Cinco is its one-click full generation of ready-to-run modeling tools (called *Cinco products*). Cinco's source languages for tool specification are restricted (i.e., domain-specific), so that lots of functionality can be generated from them, while still being quite simple. This section briefly outlines the specification formats and how they relate. A slightly more

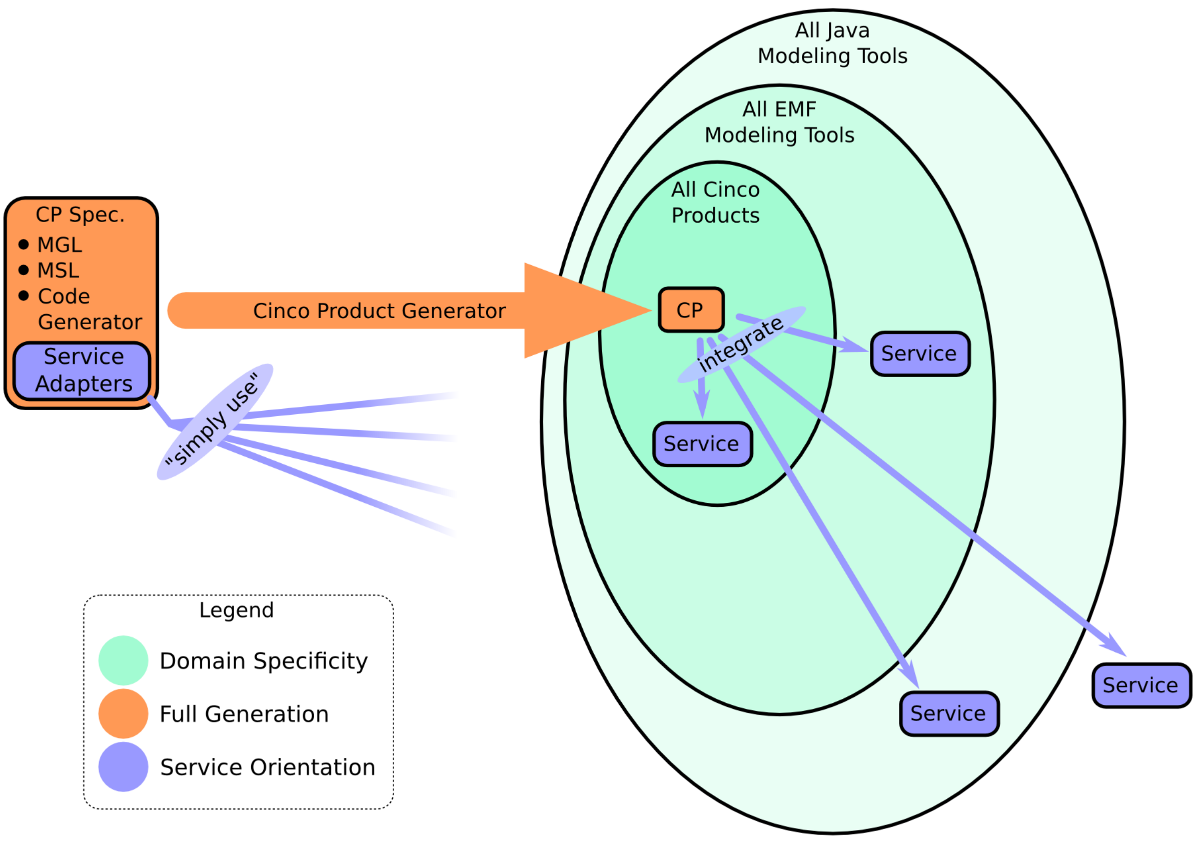

Fig. 3.1: Bird's eye view of how Cinco facilitates DFS-style development for modeling tools.

in-depth introduction is given in Sect. 3.3. For technical details and example specifications, please refer to Sects. 4–8 of [AP I: NaLyKS2017].

Fig. 3.1 visualizes the overall concept. The full specification of the Cinco product (CP) depicted on the left is generated by the Cinco product generator into the running CP. The target space on the right side illustrates Cinco's domain-specific restriction to certain kinds of modeling tools, which are a subset of all EMF-based modeling tools. These are, in turn, a subset of all modeling tools that can be implemented using plain Java. Solutions from those supersets can, however, be integrated into the CP with services, which on the specification side are 'simply used'.

Cinco's central specification format is the *meta graph language* (MGL), which roughly corresponds to metamodeling languages in other approaches. However, it is – as part of Cinco's domain-specific restrictions – a language specialized to define modeling languages with various related node and edge types. This makes it easier to define graph model structures than with other, more generic metamodeling languages. As part of Cinco's full generator, the metamodel used during the tool's runtime (which is specified with Ecore, the metamodeling language of EMF) is then generated from the MGL specification[1].

MGL is complemented by a second specification language: the *meta style language* (MSL). It defines how entity types defined in MGL should look in the graphical model and covers various forms of shapes, as well as colors, fonts, etc. Simple specifications in MGL and MSL are expressive enough to fully generate a modeling tool with a graphical editor. The Petri net [Pet66, Rei85] tool developed in

[1]Please note: while conceptually MGL is a metamodeling language, it is technically a language from which the actual runtime metamodel is generated. The concept behind this distinction is covered by the 'meta$_n$modeling' notion that will briefly be outlined as future work in Sect. 6.4.

[AP I: NaLyKS2017], for example, would require about 13,000 lines of code when manually implementing the editor using the technologies Cinco is based on (i.e., EMF, Eclipse RCP [MLA10], Graphiti [10], etc.). In contrast, MGL and MSL specifications sum up to only 41 lines of code.

MGL and MSL are both textual modeling languages developed with the Xtext framework [22] based on dedicated Ecore metamodels. In [AP III: NaNeMS2016] we discussed our plan to develop graphical variants for them, as Cinco even aims for domain experts – who are usually no programmers, and thus not so acquainted with textual specification languages – to develop modeling tools for their respective domains. Of course, variants of MGL and MSL will be developed and integrated with Cinco in a bootstrapping fashion, as it is the tool best suited for developing graphical modeling languages.

Code generators for Cinco products are primarily developed as process models using jABC with the Genesys framework. Dedicated SIBs are generated from the MGL definition that serve as an internal jABC DSL specialized on the development of code generators for exactly this language. [AP III: NaNeMS2016] furthermore envisioned generating and (reintegrating in a bootstrapping fashion) dedicated semantics languages with Cinco, e.g. for code generators or model transformations, which will be even more specialized to the individual tasks.

Beyond that, as Cinco is designed to be service-oriented itself, one can also use other technologies for code generator development. For example, in internal projects that primarily involved programmers, such as DIME [BNNS16, BFK$^+$16] and IPSUM [WMN16], we used Xtend [21] for developing code generators, as it provides a very sophisticated template mechanism. Other technologies were discussed in [AP I: NaLyKS2017] that can easily be included in a service-oriented way, e.g., model structures based on other Ecore metamodels, the ATLAS transformation language (ATL) [JABK08], OMG's Object Constraint Language (OCL) [39], or even model checking based on temporal logics. Beyond that, Java code can be included as native constructs to include arbitrary other solutions and frameworks, e.g., for graph analysis, layouting, etc.

3.2 Domain Specificity

We decided to specialize Cinco beyond common metamodeling approaches, as more domain-specific solutions can generate more functionality from smaller and simpler specifications. Of course, the term 'domain' must here be regarded on the meta level. It refers to constraining certain characteristics of the modeling tools that can be developed with the meta tool. In case of Cinco, the most important domain characteristics are:

Focus on Graph Models Cinco's metamodel specification format MGL allows for the definition of node types, edge types and container types, which directly enables the corresponding concepts of incoming/outgoing edges and node containment. The developer does not need to manually encode this information in a general entity relation schema. Also, the full generation of the corresponding Graphiti editor is primarily enabled by this focus. The constraints expressed on the metamodel are then automatically enforced by the editor, so that the user simply can not create violating models. In [AP II: SteNau2016] we utilized this feature to express the bipartiteness constraint of Petri nets as an "Archimedean Point" that ensures place and transition nodes to alternate.

Graphical Diagram Editor Without the need to know any details of underlying frameworks, the developer can define the models' visual appearance just by specifying simple shape structures. In close collaboration with the focus on graph models, full graphical diagram editors can be generated from very small specifications, greatly reducing the required effort in comparison to more general approaches.

Component Libraries Inter-model connections are restricted to Cinco's *prime references* feature (cf. Sect. 8 of [AP I: NaLyKS2017]) which supports in particular the service-oriented use of entire models as drag-and-drop components in other models and to package them to reusable and exchangeable component libraries. Cinco handles all the required management of inter-model connections in the background, so that the tool developer can very easily realize fully specifying model structures in the sense of the *one-thing approach* (OTA) [MS09].

Diagram Synchronization Many frameworks, including Graphiti used by Cinco, decouple the visual representation from the underlying model. While this is indeed a powerful feature that, for instance, enables arbitrarily many views on the same model, the management of such multiple representations is difficult to comprehend, both for the developer of the tool as well as its users. For instance, Graphiti differentiates between add/remove (i.e., changes that affect only the representation level) and create/delete (changes that affect model as well as representation). When manually applying Graphiti, this concept needs to be understood and constantly manually be adhered to, while Cinco generates the appropriate code doing it automatically in the background.

Overall, Cinco provides in particular a more holistic approach than classic solutions based on meta-modeling, as now *all* information required to generate the full-fletched graphical modeling tool can be expressed with simple specifications. The parts missing in the specification are automatically created by the code generator, as they are set by the focus on the domain characteristics. We have shown in detail in Section 10 of [AP I: NaLyKS2017] that, in contrast to Cinco, other approaches usually do not follow such a clear line of completeness and simplicity. Thus, realizing a sophisticated modeling solution with those approaches requires much more understanding on the technical level.

It could be argued that Cinco's restriction to graph structures is too severe to address a reasonable amount of cases requiring a specialized modeling tool. Of course, any model structure can in theory be encoded with it. For instance, arbitrary XML schemas can be mapped by using hierarchies of containers expressing the tree structure. So, the question of being too restricted is a matter of whether the encoding of the domain model can be done in a meaningful, intuitive way. So far, Cinco's focus on structures comprising of types of nodes, edges, and containers has proven well-suited for specialized and intuitive modeling languages. Sect. 9 of [AP I: NaLyKS2017] demonstrates this along various industrial and academic applications with very diverse model structures.

To further specialize Cinco's specification formalisms, [AP III: NaNeMS2016] sketched a plan on developing variants of jABC with Cinco, which will then replace the current jABC version as primary tool for specifying dynamic aspects of a Cinco product. Having, for instance, a specialized modeling notation for developing code generators will further enhance Cinco for the development of the 'F aspect' of DFS tools (see also Sect. 4.2).

3.3 Full Generation

Full generation of ready-to-run modeling tools can be regarded as Cinco's key feature. The basis for this generation are abstract meta-level specifications in the form of consistently interwoven models of different types. As they specify all required aspects of the Cinco products, i.e. the tools under development, Cinco provides a round-trip-free model-driven solution to the development of modeling tools.

Almost any software product can be seen as spanning two dimensions: static parts in the form of data on the one hand, and dynamic parts for data presentation, manipulation, and evaluation on the other hand. When considering modeling tools, the static parts are primarily given by the models themselves. Presentation and manipulation often comes in the form of a model editor, which is a graphical one in the context of Cinco. Evaluation of the models usually happens via model transformation, code generation, or some form of life interpretation.

Those static and dynamic aspects can be considered the actual essence of the tool. Of course, in the implementation lots of additional detail is required, e.g., concerning individual frameworks, but also considerable amounts of glue code and boilerplate code. Cinco covers all the essential aspects via different models and generates them (including all required additional code) into the complete modeling tool. Of course, as many technical details are missing in the abstract specification, they are automatically determined by the code generator. For instance, the current implementation generates the model using the Eclipse Modeling Framework (EMF) [SBPM08] and an according graphical editor with the Graphiti framework [10]. Multiple other Eclipse bundles for various parts of the product are generated in the process as well. Changing the used frameworks and their specific configurations of course requires to adapt the generator.

The individual aspects of specification have been explained in detail in Sections 4–8 of [AP I: NaLyKS2017]. Here, only a brief outline of their functionality is given:

- The *meta graph language* (MGL) is a textual meta-level modeling language. It is the basis from which the actual Cinco product's metamodel is generated. It supports the definition of *node types*, *edge types*, and how the latter may be used to connect the former. Also, *container types* and which node types may be contained can be expressed. All three of those model element types furthermore can define *attributes* of primitive types (number, text, etc.) or *complex types* also defined in the MGL.
- A second textual meta-level modeling language is the *meta style language* (MSL). It is used to define the visual appearance of the model elements defined in an MGL file. It features the definition of various shapes (like rectangle, ellipse, polygon), their colors, line styles, background images, etc., as well as text that can be either static or access an attribute of the model element. A nodeStyle or edgeStyle that is defined in an MSL file can then be assigned to a model element of the respective type in the MGL with a so-called style annotation.
- The general concept envisages a holistic model-based way. As discussed in [AP III: NaNeMS2016], this involves that dedicated specialized model types are used for defining semantics like code generation, model transformation, consistency checks, etc. The current Cinco version supports jABC processes for all those aspects. Dedicated SIBs (i.e., jABC's basic modeling components) are generated by Cinco for accessing and manipulating instances of a model type defined in the MGL. They form a domain-specific internal DSL for jABC, which is then used together with the Genesys framework [JMS08, Jö13] for modeling code generators and model transformations.
- So-called *hooks* and *actions* can be added to the MGL via annotations to provide effects on occurrence of various events in an aspect-oriented fashion. They are also implemented as processes with jABC. However, they are not primarily meant to provide semantics to models (like code generators), but to assist in the modeling. An example could be to automatically set the `number` attribute of a `State` node to the lowest unused number upon its creation by the user.

Parts of the models, especially dynamic aspects like semantic interpretations, can also be directly implemented in Java. While one reason is to facilitate service orientation with native constructs (cf. Sects. 2.3 and 3.4), another one is to support our own ongoing development in a bootstrapping fashion. One of Cinco's major design criteria is the simplified development of specialized jABC-like modeling tools. We in particular plan on developing such variants specialized to code generator development, model transformations, validation, etc., *with* Cinco *for* Cinco itself (cf. [AP III: NaNeMS2016]). This way the currently supported jABC can be stepwise removed from the Cinco ecosystem, on the one hand removing legacy technology, but on the other hand also enhancing Cinco's specification mechanisms towards a more specialized realization of the meta DFS concept. To boost this bootstrapping development Cinco generates a very comprehensible and powerful API for traversing and manipulating models. It realizes the same level of abstraction as the SIBs generated for jABC. So, it's essentially the same internal DSL for model manipulation, but with Java instead of jABC as host language.

3.4 Service Orientation

Service orientation essentially influences development along two dimensions (cf. Sect. 2.3). On the one hand, existing specification formats are enhanced with platform-level concepts (i.e. *native constructs*). They can be realized by (programming) experts who in turn do not require an understanding of the overall problem domain. On the other hand, existing *third-party* solutions can be used as libraries and flexibly be incorporated to reduce overall development effort. Service orientation abstracts from these two dimensions from the users' point of view. They just use services as existing components and do not need to know who implemented them how.

In Cinco the two dimensions are supported on three different conceptual levels:

1. extending Cinco's specification languages with native constructs
2. including whole complex solutions from the Eclipse ecosystem
3. using the generative functionality provided by meta plug-ins 'as a service'

The following subsections detail on those aspects individually.

3.4.1 Extending Specifications with Native Constructs

Service orientation with native constructs enables bridging the gap between aspects that *can* and aspects that *cannot* be expressed with Cinco's specification languages. Aiming at full model-level specification would have essentially resulted in general-purpose programming languages. The expressiveness would be high, but their utilization would also require significantly more dedicated technical knowledge on low-level concepts and frameworks. While developers of modeling tools might have some programming expertise[2], they do not necessarily want to become – or have the budget to become – experts on numerous complicated frameworks.

The ambition for maximal generality is a common problem with many existing approaches. It is in Cinco elegantly circumvented by service-oriented inclusion of native constructs without compromising the simplicity of its specification languages. This native inclusion is supported for all aspects of Cinco's tool specification:

Structure This aspect covers extending the metamodels defined with the MGL language. A very simple way of external integration is facilitated by Cinco's prime references feature (cf. Sect. 8 of [AP I: NaLyKS2017]), which on drag&drop of other models creates 'representative' nodes in MGL-defined graph models. These nodes can not only represent models from other MGL metamodels, but also arbitrary metamodels defined with Ecore. This feature can be applied for both dimensions of service orientation: on the one hand it facilitates the inclusion of models from existing third-party metamodels, for which presumably a dedicated editor already exists that can be used to create those models. On the other hand, this native inclusion of Ecore metamodels can

[2]Although Cinco even aims at enabling domain experts to develop their modeling tools themselves, the more common case will probably be that programmers without dedicated knowledge on frameworks for modeling language development are confronted with the task.

be used for the realization of features not supported by Cinco's MGL, e.g., for a model that is not reasonably expressed as a graph. In the projects we have done with Cinco so far this possibility was mainly used for small configuration models that are more conveniently edited with very simple text or dedicated forms. Their uncomplicated nature made them easy enough to develop with EMF's core features.

Model Presentation There are basically two native ways to extend the MSL-defined visual aspects of models: one is to provide the implementation of the underlying platform, which is our usual approach for native constructs. In case of the graph modeling framework Graphiti used by Cinco, this approach is highly intricate and too difficult to handle. We thus decided against that kind of native inclusion for Cinco. Another, much more simple and effective, way is to just allow the inclusion of image files that determine the appearance of a node (or part of it). While currently only raster images are possible, a good option for the future could be the inclusion of vector graphics (e.g. SVG files). Like Cinco's MSL, they have an inherent awareness of elements, shapes, paths, curves, layering etc., but allow for much more visual possibilities than MSL. Also, a variety of editors is freely available.

Dynamic Aspects The dynamic aspects of a Cinco product primarily comprise semantics definitions like code generators and transformations, but also the aforementioned assisting hooks and actions. They all have in common that they can either be defined with jABC process models or with native Java code. Obviously, the latter directly allows for the inclusion of arbitrary third-party services, but is not necessarily the most simplicity-driven way. On the other hand, one of jABC's most fundamental concepts is the service-oriented thinking in terms of ready-to-use components (i.e., SIBs). Arbitrary services (such as search or traverse algorithms, calculations and statistics, layouting, etc.) can thus easily be integrated without the need to know how they are realized. Even handling of the model structures is simplified by dedicated SIB libraries generated from the MGL specification. Summarizing, for the definition of all dynamic aspects Cinco can directly rely on the extensive support for service orientation provided by jABC.

3.4.2 Framework Level

A different form of service orientation is provided by the native inclusion of functionality directly on the framework level. As Cinco is based on the Eclipse Rich-Client Platform (RCP) [MLA10], arbitrary RCP bundles can be included in a Cinco product. This is done by just declaring a bundle's unique identifier as a dependency within the Cinco product definition. It similarly works for so-called features, which essentially subsume a set of bundles under a new name.

On the one hand, this solution allows one to enrich the modeling tool with additional manually written 'native' aspects, such as views, wizards, etc. But more importantly, it enables the inclusion of existing Eclipse bundles as third-party services. This opens up a wealth of opportunities into the modeling tool, ranging from full IDEs for Java or C (via the according bundles/features for *Java development tools* (JDT) [4] or *C/C++ development tooling* (CDT) [3]) to the inclusion of source code management for Git or Subversion.

3.4.3 Meta Plug-ins

An approach to meta-level service orientation is provided by special Cinco extensions called *meta plug-ins*. They provide a way of enhancing the generative functionality of Cinco: services are not just called by the product or integrated 'as is', but specifically generated based on the actual specifications of the Cinco product (MGL, MSL, etc.).

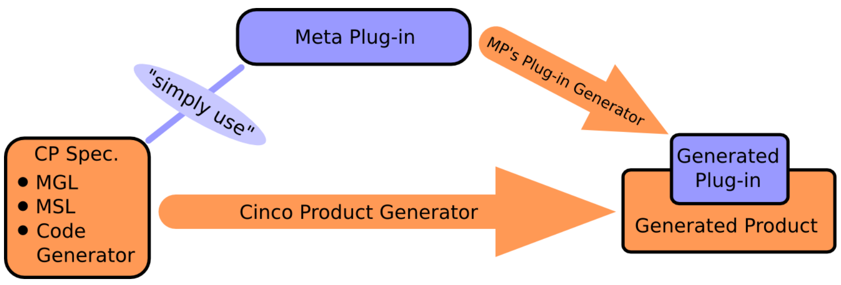

Fig. 3.2: Meta plug-ins contribute to the Cinco product generator and provide a generative way of service orientation.

Meta plug-ins contribute features potentially useful for many modeling environments and integrate those into the generated tool as a service. Examples for such features are model checking, reachability analysis, path finding, layouting, etc. For models targeting some form of control flow semantics (processes, workflows, etc.) even more specialized meta plug-ins are possible, e.g. for code generation, execution or data-flow analysis. Essentially, the Cinco product developer activates a meta plug-in by just adding the corresponding annotation into the MGL file. Often, additional parameters specific to this meta plug-in can be set to further influence the generated service.

Fig. 3.2 sketches the approach. A meta plug-in is ‘simply used’ on the Cinco product specification side, just like other services (cf. also Fig. 3.1). During the product generation, the meta plug-in generates dedicated functionality tailored towards the overall specifications. This functionality is integrated as a plug-in in the generated product. Fig. 6 of [AP I: NaLyKS2017] and Fig. 4 of [AP II: SteNau2016] (as well as the corresponding text passages) provide more details on how meta plug-ins contribute to the overall concept. They in particular include the aspect that not the complete plug-in is generated, but accesses some runtime API also provided by the meta plug-in.

While meta plug-ins are primarily meant to be provided as third-party extensions, they can of course also be implemented by a Cinco product developer. This is particularly useful, if a family or product line of modeling tools is developed rather than a single instance, as common concepts can be shared between members of the family.

The concept of meta plug-ins furthermore plays a crucial role in the continuous improvement of the whole Cinco ecosystem: solutions initially integrated in the classic service-oriented fashion for a single Cinco product (as described in Sect. 3.4.1) can later be generalized to become meta plug-ins, and thus contribute their functionality to a wide range of generated tools. Three examples for this meta-level generalization, which originate from the DIME project, are presented as future work in Sect. 6.2. Overall, meta plug-ins are a key factor for a pragmatic approach towards a meta unification for specialized development tools as discussed in [AP III: NaNeMS2016].

Technically, a meta plug-in generates one or more plug-ins for the developed Cinco product. The work presented in [AP V: NNLSJM2013] laid the foundation into that direction by presenting a concept to generate plug-ins for the jABC framework. We plan on integrating the synthesis-based automatic completion of underspecified processes utilized in [AP V: NNLSJM2013] (i.e., in the context of jABC done with the PROPHETS framework) also for meta plug-ins in Cinco. This will open up more flexible ways of parametrizing a meta plug-in, as temporal-logic constraints (or more user-friendly templates) can be used to express the desired intent instead of implementing all the different behaviors that might arise from user-given parameters.

4 CINCO: Support for DFS Tool Development

Basically, any tool development framework can be used to implement the DFS properties. It is the aspect of simplicity that is usually missing (cf. Sect. 10 of [AP I: NaLyKS2017]). But a more specialized framework can aid in various forms, and CINCO is specifically designed to provide extensive support for all three properties.

Whether or not a developed modeling tool fulfills the DFS properties is primarily a design philosophy. They are not enforced in any way. If the tool developer decides against one (or more) of them, CINCO is still a good choice. As discussed in Chap. 3, it is simple and efficient to apply due to being itself a DFS tool.

Many of the aspects discussed in this chapter have already been covered before, as they directly stem from CINCO's specification languages presented in Sect. 3.3 and the service-oriented inclusion introduced in Sect. 3.4. The shorter presentation in this chapter thus serves as a different perspective.

4.1 Domain Specificity

Central idea of any development framework that incorporates metamodeling is to establish domain specificity in the developed modeling tool. However, in CINCO, the combination of metamodel (MGL language) and visual definition (MSL language) facilitates a very simple way of generating the structure of the domain model together with an according graphical editor.

In addition to the easy definition of a domain model with the MGL language, CINCO's prime references (cf. Section 8 of [AP I: NaLyKS2017]) provide a simplicity-driven way of separating the target domain into individual aspects with dedicated visual languages that together facilitate interconnected domain models. Essentially, the idea is to simply drag&drop a model into another model, so that a new node is created in the second model *representing* the first model. Of course, this connection must be explicitly allowed within the MGL specifications. With such representative nodes (heterogeneous) hierarchical structures are immediately possible. All code required in the modeling tool for data handling, linking and instantiation is generated by CINCO. Thus, tool developers can very easily provide their tool users with an intuitive way of constructing consistently interconnected models able to cover all aspects of the target domain.

4.2 Full Generation

A base premise for fully generating the complete application is that all its aspects must be captured in models. This is mainly covered by the techniques used for defining distributed heterogeneous domain model structures presented in the previous section.

Above that, the development of code generators can be simplified. Much of this simplification in Cinco is directly provided by the model-based development of code generators with the Genesys framework and jABC. Thus, this section primarily focuses on further dedicated enhancements provided for this approach. Of course, Cinco's flexible service-oriented extensibility (cf. Sect. 3.4) allows any tool set that can handle Ecore models to be utilized for code generation in a Cinco product. Those external frameworks even profit from the generated API which is the same the jABC SIB libraries are based on and which, for instance, abstract from the distinction of model and representation.

4.2.1 Traversing the Model

When generating code from a model – or a complex structure comprised of multiple models – the generator needs to analyze and (stepwise) process it. For this model traversal, an API is usually provided by the framework. In Cinco this API is generated from the specifications made in MGL. It is the basis for dedicated jABC SIBs that are then used to traverse the model for code generation with Genesys. Above that, the API can also be used directly in code or by other code generation frameworks from the Eclipse modeling context. In contrast to existing approaches, in particular Cinco's specialization to the domain of graph models allows for several improvements and simplifications.

The API comprises dedicated components to handle the graph model-specific concepts of predecessors and successors for nodes, incoming and outgoing edges for nodes, as well as source and target nodes for edges. Cinco's analysis of the MGL ensures that only valid options are provided in the API. This also makes the API correctly typed, so that low-level technical concepts like 'class casting' are not required. The Petri net examples from [AP I: NaLyKS2017] and [AP II: SteNau2016] made use of exactly this feature. The 'Archimedean Point' of bipartiteness implies that all successors of a place are transitions and vice versa, making the traversal really simple.

Similar to these node/edge relations, the API provides components to traverse the containment structure of nodes. Dedicated container nodes can again contain other nodes (and containers), so that models with local hierarchy (e.g. for sub-states in a state chart) can be built. Beyond that, access to prime referenced models is also easily possible via this API. All code required for handling is generated by Cinco.

Of course, designing such an API is possible with any metamodeling framework, or even directly using code. But with Cinco, for each developed modeling tool it is automatically generated from the information provided within the MGL, so that no additional work is required for the tool developer.

4.2.2 Producing Code

While traversing a model structure, the actual code is generated. Template mechanisms are usually used to cover arbitrary textual output. A template is a piece of output code that contains certain placeholders, which are dynamically filled with information from the model. This part is highly specific to the developed modeling tool, as it very much depends on the model structure as well as the target language.

While many template languages mix traversal code and generated output, Genesys provides a clean distinction between those two. Traversal is done in jABC process models using dedicated SIB libraries. Templates, on the other hand, parameterize dedicated output SIBs. There exist libraries for various template languages, such as Apache Velocity [23] or StringTemplate [42]. The service-oriented nature of jABC allows to extend those easily.

As sketched in [AP III: NaNeMS2016], we plan on using Cinco to generate a process modeling tool based on the jABC concepts, but specialized on code generation. Here, for instance, we can put more focus on the template aspect, which is currently hidden in SIB parameters. Furthermore, a common problem of template mechanisms is that the output is just recognized as plaintext. Errors (like syntactical ones or missing classes etc.) are only revealed after code generation. With specializations to dedicated target languages (Java, C++, Python, etc.), we can provide syntax highlighting within the templates, or even some basic IDE-like support for navigating into classes, refactoring, etc.

4.2.3 Other Target Formats

The DFS concept is not limited to generating textual source code from models. Full generation also comprises generating other target space artifacts. This in particular incorporates creating other models, which then can be further processed (interpreted, generated, or transformed).

For such model-to-model transformations the integration of jABC into the Cinco landscape also provides different levels of aid. If the target model type is part of a Cinco product, i.e., based on an MGL, the same API generated for traversing the model structures can be used, as it also provides components for creation and modification of according models.

Arbitrary Ecore metamodels are also supported. The TransEM [Lyb12] framework, which is an extension to Genesys' Ecore integration [Jö13, Ch. 7] generates model transformation SIBs from any Ecore metamodel.

Again, the service-oriented nature of jABC allows for the inclusion of additional functionality for the creation of arbitrary other target artifacts. For instance a library to create XML files conforming to a dedicated XML schema could be provided – and even be automatically created from the schema itself.

4.3 Service Orientation

Prime references are Cinco's major feature to facilitate service orientation in a Cinco product. The basic idea of 'just using' drag&drop components essentially was created as a generalization of jABC's SIB concept. The aspect of third-party components is directly available for the tool developer, as any model for which a prime referencing node is defined can directly be used. Of course, the code generator needs to implement the corresponding behavior (e.g., as a service call) when encountering such a node during traversal.

For native constructs, using dedicated adapter model types has proven to be an adequate strategy. Corresponding adapter models then either contain all information required to call a native third-party service, or include the native code itself. Using those adapter models as services is then again enabled by the prime references feature.

5 Related Approaches

Sect. 10 of [AP I: NaLyKS2017] compares CINCO with seven 'competing' development frameworks for graphical modeling tools. Focus for this comparison is the simplicity of specification formats and overall accessibility. The predominant result is that the majority of regarded features is rather complicated in almost all the other frameworks.

This chapter first gives a short overview on these frameworks with references to according publications. Then, to complement the analysis from [AP I: NaLyKS2017], it discusses to what extent these frameworks support the DFS properties on the meta level, i.e., whether or not they can themselves be regarded as DFS tools. This provides a comparison to CINCO's unique characteristic of being a DFS tool specialized to the domain of modeling tool development, as presented in Chap. 3.

As generally the development of DFS tools is possible with all those frameworks, the analysis of simplicity done in [AP I: NaLyKS2017] almost directly corresponds to the amount of dedicated aid provided for this development. Therefore, this aspect, which corresponds to CINCO's support for DFS tool development presented in Chap. 4, is not discussed again in this chapter. Summarizing, they all provide (usually less simple) ways of domain specificity through metamodels as well as dedicated ways of code generator development. Above that, in particular service orientation – i.e., ready-to-use component libraries that are easily possible in CINCO using prime references – is a highly complicated aspect for almost all tools.

5.1 Overview of Frameworks

The compared frameworks can roughly be classified into two groups. On the one hand, several solutions are based on Eclipse technology. *Epsilon* [KRGDP15, KRbA$^+$10] [7, 8] comprises several model specification, query, and transformation languages. Part of this framework is EuGENia, which generates GMF [9] editors based on Ecore metamodels enriched with specific annotations. *Spray* [16] also focuses on the generation of graphical modeling tools for existing Ecore metamodels. It provides a dedicated DSL to specify graphical editors that are then generated for the Graphiti API. *Sirius* [6] is a recent addition to the Eclipse Modeling Project [5]. Again, for a given Ecore metamodel a tool developer can specify so-called modeling workbenches utilizing Sirius' declarative specification languages, providing graphical, table, or tree visualizations and editors for EMF models. *Marama* [GHL$^+$13] [13] is also based on Eclipse (unlike its predecessor Pounamu [ZGH04]), but takes a more holistic approach towards the generation of graphical modeling tools. Independent of the EMF metamodeling framework, it realizes a complete set of visual declarative specification languages to define all aspects of the generated tool.

On the other hand, several realizations based on completely different underlying technologies were compared. *MetaEdit+* [KLR96] [14] is a complex and mature stand-alone tool suite functioning as the reference implementation of *domain-specific modeling* (DSM) [KT08]. Similarly, the *generic modeling environment* (GME) [LMB$^+$01, LMV03] provides the generation of visual model editors from metamodel specifications. More recently, the GME group has also developed *WebGME* [20], a cloud-based environment which circumvents installation effort and platform restrictions by running in the web browser. However, many central aspects of the meta level, i.e., the level of DSL development, still have to be performed on the server side. *DEViL* [KPJ98, SCK08] relies on multiple specification and compiler frameworks and essentially generates visual structural editors from declarative specifications in various textual formats.

5.2 Discussion on Meta DFS Capabilities

None of the compared frameworks actually promotes a sense of service orientation like Cinco does. This is a remarkable observation, as in over 20 years of MetaFrame tools and jABC tools the concept has proven to be an extremely intuitive way to facilitate flexible process modeling. Its mechanism of SIB libraries can even be utilized to provide a form of domain specificity. Some of the compared frameworks do indeed allow to include native code (e.g., Sirius or GME), and, of course, all EMF-based approaches can somehow be accessed with other Eclipse solutions. But as they do not follow the mindset of 'simply using services', those solutions are usually much more complicated.

Furthermore, only GME and MetaEdit+ provide a considerable degree of domain specificity comparable to Cinco's restriction to graph model structures, but they are less focused on simplicity. Other approaches are not as holistic and sometimes only impose restrictions within parts of the specifications. The Spray framework, for instance, is a generative approach for Graphiti diagram editors (and thus, like Cinco, focuses on graph structures). But it is based on Ecore metamodels, which are more complicated and more generic. Thus, Ecore is not fully covered, whereas the domain-specific restriction in Cinco is reflected in both the MGL metamodeling language as well as the MSL visual language.

All compared frameworks can, however, essentially be regarded as following a fully generative approach. This is not really surprising, as it was one major selection criterion for the comparison. Few provide a true one-click solution like Cinco, though. Especially the approaches extending EMF and Ecore provide no seamless workflow here, as they are still independently developed. But all do indeed conceptually prohibit modifying generated code.

Considering 'full generation' might seem contradictory for approaches with interpreted specifications, which is in particular done by the ones facilitating live editing[1] (Sirius, MetaEdit+, Marama, WebGME). However, in the sense of complete specifications that do not need a round-trip, they can be considered equivalent[2].

In contrast to the fully generative approaches, consider, for example, the widespread Eclipse GMF framework: from a base specification, i.e., an annotated Ecore metamodel, various specification models are generated in multiple steps. In each of those steps the developer has to add further information to the

[1] Live editing means that within a single tool changes made to the metamodel specifications are immediately reflected in the modeling. A major drawback is that this usually makes the specifications more complex and that no clean distinction between meta levels is maintained.

[2] Full generation as defined in Sect. 2.2 requires that a specification is complete and can stepwise be fully automatically translated into something executable. If the specification is executable on its own, this requirement is trivially met with zero steps.

current model. There is some automatic mechanism that tries to facilitate round-trip by recognizing and maintaining changed parts, but this rarely works well. That is why it was not included in the comparison in [AP I: NaLyKS2017] and probably also why approaches like Epsilon's EuGENia emerged, which fully generates all required GMF specification models.

In summary, the concept of full generation is not generally a unique feature. However, the aspect of simplicity is often neglected, which in Cinco is enabled by the addition of domain specificity and service orientation.

6 Future Work

This chapter indicates some directions for future work in the context of CINCO and DFS (meta) tools. A large amount of features, fixes, and enhancements for CINCO has already been discussed, planned and partly realized. Some of them are briefly outlined in Sect. 6.1.

However, the main focus of this chapter lies more on the long-term conceptual enhancements:

- Sect. 6.2 presents the concept of generalizing solutions from single projects to the meta level, which is done along some examples from the DIME project, the so far most ambitious tool realized with CINCO.
- Sect. 6.3 discusses the idea of extending CINCO in a bootstrapping fashion with CINCO-generated modeling solutions.
- Sect. 6.4 introduces 'meta$_n$modeling', which aims at providing a notion that combines the commonly used metamodel hierarchy with domain specialization and chaining of (full) generators.

6.1 CINCO Enhancements

Many ideas for enhancements evolved during development and use of CINCO. This section briefly outlines two particularly interesting and promising ones that can further advance CINCO regarding the provision of a truly holistic approach to the development of DFS tools.

6.1.1 Synchronous Textual and Graphical Syntax

A recent addition to CINCO is a feature called 'GraText' (combining 'graph' and 'text'). It replaces the XML-based serialization of the underlying EMF framework by providing dedicated textual syntaxes for all CINCO product model types. This is done by automatically generating MGL-specific Xtext grammars. GraText also provides a synchronized multi-page editor that allows one to seamlessly switch between this textual and the normal graphical representations of a model. The role of GraText is twofold: on the one hand it can be used by users preferring a textual syntax (while other users still can use the graphical syntax), on the other hand it decouples the intellectual property expressed in models (e.g., machine setups, business processes, etc.) from the actual technical realization of the underlying metamodeling framework and its serialization format.

So far, the generation of the textual syntax for a given MGL follows a generic generative pattern. We plan on enhancing this by introducing the (optional) provision of grammar snippets that can be specified for nodes and edges. This will allow for a more specialized syntax, but without the added complexity of specifying the full grammar manually, as the overall structure will still be managed on the generative level. Basically, this feature can be regarded as a textual equivalent to the MSL styles we define for our graphical editor.

6.1.2 Semantic Three-Way Merge of Models

Developing with Cinco usually means that the resulting DFS tool is meant to be used for software development again. It is thus to be expected that the development artifacts (i.e., the models) need to be managed in some kind of versioning system like Subversion or Git. However, those systems are optimized for source code, for which the provided three-way merge[1] usually works quite satisfactory. Even when models are persisted using the textual GraText representation, a more reliable mechanism is required.

The *model compare and merge* framework (MCaM) [Wir15] was developed to add sophisticated merging capabilities for jABC process models. The framework itself is general, though. We thus plan to develop a meta plug-in that generates MCaM merge handlers specialized to the individual MGLs. This would allow changes to be captured semantically (e.g., ‘added node X’, ‘reconnected edge A from node X to node Y’, etc.) and facilitate a domain-specific automatic merge for any tool developed with Cinco.

6.2 Meta-Level Generalization

The *DyWA integrated modeling environment* (DIME) [BNNS16, BFK$^+$16] is one of the most sophisticated DFS tools developed with Cinco so far. It is a development environment for the full model-based specification and generation of multi-user web applications (cf. Sect. 9.2 of [AP I: NaLyKS2017] for a brief summary).

The DIME project significantly drove the advancement of Cinco in the past year and various valuable insights have been gained from it for new Cinco functionality. In many regards, parts of DIME may serve as prototype implementations for new Cinco core features or meta plug-ins to specifically generate them for a wide range of tools in a service-oriented way (as described in Section 3.4.3).

6.2.1 Executable Processes

One major driving force of Cinco was the goal to develop more specialized successors for the jABC framework. With the DIME project we have shown that Cinco is very well capable of that task. DIME provides multiple variants of process models specialized to the domain of web applications. Moreover, we also realized completely different model types for data structures as well as user interfaces, which are all combined to one overall specifying model structure and generated to be fully executable.

Introducing the concept of processes to the level of Cinco specifications would allow tool developers to simply add ‘executability’ to their modeling tools, while Cinco could generate specific adaptations for most of the code we had to manually write for DIME. Sect. 4.2 of [AP III: NaNeMS2016] presents a more detailed perspective on this concept of meta-level generalization.

[1]Three-way merge is required when two developers make changes based on the same old version of a file, as then the new file can not just replace the old one.

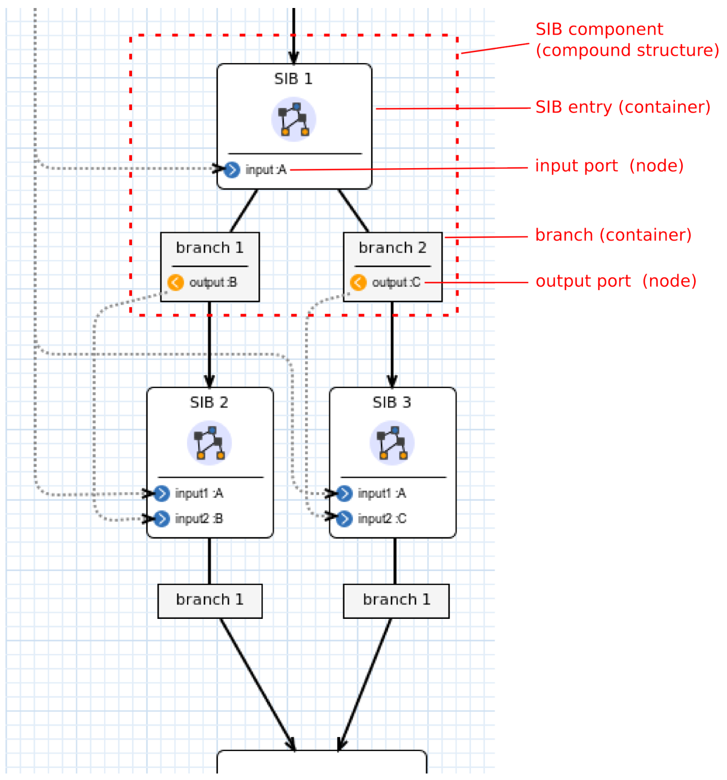

Fig. 6.1: A DIME SIB component comprises of various Cinco containers and nodes.

6.2.2 Compound Structures

The concept for executable process components in DIME was adopted from jABC's SIBs. A SIB can have multiple *data inputs* as well as multiple *result branches*, which again can have multiple *data outputs*. While the branches define the outcome of the SIB execution, and thus the control flow, inputs and outputs define the data flow between SIBs.

A major difference between SIBs in jABC and in DIME is, that, in addition to control flow, the latter also represents the data flow visually. Inputs and outputs are visible 'ports' on SIBs and branches, which are connected with dedicated data flow edges. In terms of Cinco's meta-level specifications, SIBs and branches are containers that hold input port nodes and output port nodes, respectively. Thus, while a SIB feels like one service-oriented component for a DIME user, it is actually realized as several components on the specification level of Cinco (cf. Fig. 6.1).

Currently, this structure is manually maintained in various places in DIME's implementation. For instance, as a SIB can represent a whole process in hierarchical model structures, the API (i.e., branches and ports) needs to be kept in synchrony: if the API of a process changes, it needs to be validated (or even updated) in the SIBs representing this process.

To cover this kind of pattern and behavior in a generative way, we plan on introducing a concept of 'compound structures' on the meta level. These structures will allow tools to have a more abstract notion of components consisting of multiple nodes and containers.

6.2.3 Component Views

Models of the three DIME modeling languages (processes, data, and user interfaces) are frequently interconnected using Cinco's prime references feature. However, to provide specific presentations of the three different ‘kinds of libraries’, we realized dedicated views showing the usable elements in a tree structure. Data types, for instance, are arranged according to their inheritance relation, whereas for processes the arrangement corresponds to the project’s directory structure. Even service taxonomies following jABC’s concept of domain modeling are planned for the future.

Those three views are currently implemented manually, but, although we reuse lots of functionality by inheritance, significant portions of code are very similar. We thus plan to add a meta plug-in to Cinco that generates such dedicated component views according to some simple specification, e.g., which elements to display and how to arrange them in said tree structure.

6.3 Bootstrapping Cinco

As indicated in several passages of Chaps. 3 and 4, we plan on replacing jABC as the current way of specifying the semantics of a modeling language by variants more specialized to the individual tasks of code generation, model transformation, validation, etc. However, the idea is not limited to the modeling of semantics. Of the other approaches compared in Sect. 5, some provide graphical languages for metamodel specifications as well as visual appearances. In particular for the latter, which is given by MSL definitions in Cinco, this surely is a more intuitive way.

Of course, we will use Cinco to generate all those dedicated graphical specification languages and then feed them back into Cinco itself. This enhancement in a bootstrapping fashion is described in [AP III: NaNeMS2016].

As many of those languages will provide process models again, the approach will in particular profit from the meta-level generalization of the concept of ‘executable processes’, as introduced in Sect. 6.2.1.

6.4 Meta$_n$modeling

An essential aspect of many metamodeling formalisms is a concept commonly denoted as ‘metamodel hierarchy’. In this hierarchy, a model type specification, i.e., which elements a conforming model consists of and how they may relate, is defined with a metamodel. This metamodel in turn is based on a dedicated specification format itself, which is then the metamodel of the metamodel, and the meta-metamodel of the model[2]. This is where special metamodeling frameworks usually come into play. They provide support to define various metamodels using a metamodeling language, so that model types defined this way comply with certain conventions of the framework. This allows one to automatically generate code for handling these models.

However, this notion is not flexible enough for a full generative approach like Cinco.

For example, consider a Petri net modeling tool implemented with the EMF metamodeling framework. A modeled Petri net – e.g., the dining philosophers – then corresponds to the Petri net metamodel. In other words, the Petri net Ecore specification is the metamodel of the philosophers model. In turn, Ecore itself is the metamodel of the Petri net Ecore specification and thus the meta-metamodel of the philosophers model.

But what if the Petri net metamodel is itself generated from some other specification, which is exactly what Cinco does based on the MGL language? A concept is needed to describe the relation between the philosophers model and the Petri net MGL specification. In some sense, the Petri net MGL is the

[2]The concept of terminating this (in theory) infinite chain of meta levels by using a reflexive metamodeling language able to describe itself is not required in this section, and thus not further explained.

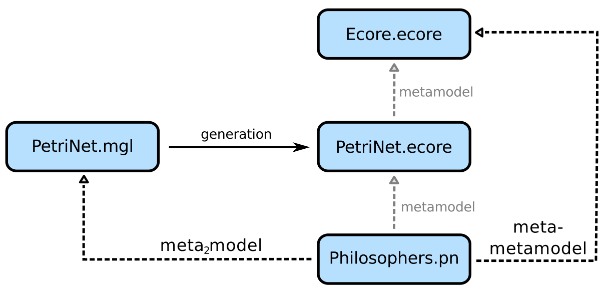

Fig. 6.2: Illustration of meta-metamodel vs. meta$_2$model

metamodel, but so is the Ecore generated from it. Also, chaining 'metas' is already reserved for the metamodel of the metamodel, but the Petri net MGL is clearly not the metamodel of the Petri net Ecore.

For such cases, I suggest a notion of 'meta$_n$modeling'. More concretely, here the Petri net MGL would be the meta$_2$model of the philosophers model and the Petri net Ecore would be the meta$_1$model[3]. Fig. 6.2 illustrates the distinction between this meta$_2$model and the commonly used notion of meta-metamodel.

As Cinco is itself a DFS tool, the whole Cinco product is generated from a consistent set of specification files, such as models, resources, Java source code, constraints, etc. As just discussed, this generative approach particularly includes that the product's metamodel is generated. However, it is important to note that not only this metamodel is generated, but everything else that can be based on the information fixed by the provided specification combined with Cinco's own domain specialization. Just like in common metamodeling, where the structure of data is known, so that event handlers or undo/redo command stacks can be generated automatically, we know here that the data structure is a specifically visualized graph model, so that the whole graphical editor can be generated.

Thus, the higher n in meta$_n$modeling not only establishes a relation between models and generated metamodels, but further specializes the whole context. In the course of projects done with Cinco, we already encountered several potential cases of meta$_3$modeling:

- The specialized languages transformed into the PSM superset language, as presented in [AP IV: NaTISL2014], could actually be generated from a higher-level model that defines which of the language features like time, probability etc. need to be included. The code generators would still be based on the PSM superset language, but the transformations could easily be generated. The idea of formalizing the notion of meta$_n$modeling actually stems from this PSM study.
- Similar to PSM, various variants of Petri net modeling tools could be generated from a 'Petri net tool language' more specialized than MGL. The inclusion of features like colors, capacities, arc weights, time etc. would then be based on this specification and realized by a code generator.
- It might even be interesting to explore, whether realizing the meta generalization of the 'executable processes' concept, as discussed in Sect. 6.2.1, can be done with a dedicated meta$_3$modeling language, so that we actually generate MGLs, MSLs etc. in the fashion of DIME.

In all those cases, according code generators need to generate all specifications required on the meta$_2$modeling level, so that the overall approach remains fully generative. Of course, the aspect of service orientation can then again be used to include into the meta$_3$modeling level 'native' specifications of the meta$_2$modeling level[4]. In general, fleshing out a good conceptual foundation and facilitating sophisticated tool support for such endeavors provides a major challenge for future research.

[3]Omitting the $_1$ (like a factor or exponent) then even sustains the common notion of metamodel.

[4]Generally, any meta$_j$modeling level can be extended with service-oriented inclusion of 'native' artifacts from all meta$_i$modeling levels with $i < j$.

7 Conclusion

This thesis presented a novel approach to model-driven software engineering that focuses on simplicity through highly specialized tools. Domain experts are provided with development tools tailored to their individual needs, where they can easily specify the intent of the software using their known terms and concepts. This domain specificity (D) is combined with full generation (F) and service orientation (S) to facilitate pushbutton generation of the entire software. The DFS combination counteracts many common problems from existing model-driven approaches, and thus has the potential of being more widely accepted by practitioners.

Key to the approach is a holistic solution that in particular also covers the simplicity-driven development of according DFS modeling tools. This simplicity is achieved by self-application of the concept on the meta level: generative domain-specific modeling tools are fully generated from models and services specialized to the (meta) domain of modeling tools.

The Cinco meta tooling suite has been presented as a corresponding meta DFS tool focused on the development of graphical modeling tools for graph structures. It has been very successfully applied to numerous industrial and academic projects, and thus also serves as a proof of concept for the DFS approach itself. In particular in the context of DIME, Cinco has demonstrated the ease of development for tools fully supporting all three DFS factors. DIME, being a service-oriented process modeling tool specialized to the domain of web applications, demonstrates that Cinco is very well capable of developing more specialized variants of jABC.

The unique combination of the three DFS strategies and Cinco's meta-level approach towards their realization in practice paved the way towards a paradigm change in model-based domain-specific software engineering that is strongly focused on simplicity. Overall, specialized generative approaches impose levers that simply can't be achieved by manual development. Conceptual generalization of solutions from single projects as well as extending Cinco in a bootstrapping fashion with Cinco-generated modeling languages will be next steps towards a meta-level common conceptual core that drives this concept even further.

Also, fleshing out the aspect of meta$_n$modeling as a conceptual foundation for stepwise generation into lesser domain-specific DFS solutions seems particularly promising for future research, as generating into an '$n - 1$' modeling language is naturally simpler than into lower levels (e.g., directly code). While this could open up a whole new dimension of simplicity, developing sophisticated tool support will be an interesting challenge.

References

[AFG+09] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, Electrical Engineering and Computer Sciences. University of California at Berkeley, 02 2009.

[BCW12] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool, 2012.

[BDG+15] Agata Berg, Cedric Perez Donfack, Julian Gaedecke, Eike Ogkler, Steffen Plate, Katharina Schamber, David Schmidt, Yasin Sönmez, Florian Treinat, Jan Weckwerth, Patrick Wolf, and Philip Zweihoff. PG 582 - Industrial Programming by Example. Technical report, TU Dortmund, 2015.

[BFK+16] Steve Boßelmann, Markus Frohme, Dawid Kopetzki, Michael Lybecait, Stefan Naujokat, Johannes Neubauer, Dominic Wirkner, Philip Zweihoff, and Bernhard Steffen. DIME: A Programming-Less Modeling Environment for Web Applications. In *Proc. of the 7th Int. Symp. on Leveraging Applications of Formal Methods, Verification and Validation, Part II (ISoLA 2016)*, volume 9953 of *LNCS*, pages 809–832. Springer, 2016.

[BHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web - A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, 284(5):34–43, May 2001.

[BNNS16] Steve Boßelmann, Johannes Neubauer, Stefan Naujokat, and Bernhard Steffen. Model-Driven Design of Secure High Assurance Systems: An Introduction to the Open Platform from the User Perspective. In T.Margaria and Ashu M.G.Solo, editors, *The 2016 International Conference on Security and Management (SAM 2016). Special Track "End-to-end Security and Cybersecurity: from the Hardware to Application"*, pages 145–151. CREA Press, 2016.

[DP06] Brian Dobing and Jeffrey Parsons. How UML Is Used. *Communications of the ACM*, 49(5):109–113, 5 2006.

[DS12] Markus Doedt and Bernhard Steffen. An Evaluation of Service Integration Approaches of Business Process Management Systems. In *Proc. of the 35th Annual IEEE Software Engineering Workshop (SEW 2012)*, 2012.

[Erl07] Thomas Erl. *SOA: Principles of Service Design*. Prentice Hall, 2007.

[FBL10] Andrew Forward, Omar Badreddin, and Timothy C. Lethbridge. Umple: Towards Combining Model Driven with Prototype Driven System Development. In *Proc. of the 21st IEEE Int. Symp. on Rapid System Protyping (RSP 2010)*. IEEE, 2010.

[FBLS12] Andrew Forward, Omar Badreddin, Timothy C. Lethbridge, and Julian Solano. Model-driven Rapid Prototyping with Umple. *Software: Practice and Experience*, 42(7):781–797, 2012.

[Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[FL08] Andrew Forward and Timothy C. Lethbridge. Problems and opportunities for model-centric versus code-centric software development: a survey of software professionals. In *Proc. of the 2008 Int. Workshop on Models in Software Engineering (MiSE 2008)*. ACM, 2008.

[Fow00] Martin Fowler. Continuous Integration, 9 2000. Aktualisierte Version vom Mai 2006.

[FP11] Martin Fowler and Rebecca Parsons. *Domain-specific languages*. Addison-Wesley / ACM Press, 2011.

[GHL$^+$13] John Grundy, John Hosking, Karen Na Li, Norhayati Mohd Ali, Jun Huh, and Richard Lei Li. Generating Domain-Specific Visual Language Tools from Abstract Visual Specifications. *IEEE Transactions on Software Engineering*, 39(4):487–515, 2013.

[HS05] Michael N. Huhns and Munindar P. Singh. Service-Oriented Computing: Key Concepts and Principles. *IEEE Internet Computing*, 9:75–81, January 2005.

[JABK08] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. ATL: A model transformation tool. *Science of Computer Programming*, 72(1-2):31–39, 2008.

[JLM$^+$16] Sven Jörges, Anna-Lena Lamprecht, Tiziana Margaria, Stefan Naujokat, and Bernhard Steffen. Synthesis from a Practical Perspective. In *Proc. of the 7th Int. Symp. on Leveraging Applications of Formal Methods, Verification and Validation, Part I (ISoLA 2016)*, volume 9952 of *LNCS*, pages 282–302. Springer, 2016.

[JMS08] Sven Jörges, Tiziana Margaria, and Bernhard Steffen. Genesys: service-oriented construction of property conform code generators. *Innovations in Systems and Software Engineering*, 4(4):361–384, 2008.

[Jö13] Sven Jörges. *Construction and Evolution of Code Generators - A Model-Driven and Service-Oriented Approach*, volume 7747 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Germany, 2013.

[KJMS09] Christian Kubczak, Sven Jörges, Tiziana Margaria, and Bernhard Steffen. eXtreme Model-Driven Design with jABC. In *CTIT Proc. of the Tools and Consultancy Track of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA)*, volume WP09-12, pages 78–99, 2009.

[KLR96] Steven Kelly, Kalle Lyytinen, and Matti Rossi. MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. In *CAiSE*, volume 1080 of *Lecture Notes in Computer Science*, pages 1–21. Springer Berlin / Heidelberg, 1996.

[KMSN07] Christian Kubczak, Tiziana Margaria, Bernhard Steffen, and Stefan Naujokat. Service-oriented Mediation with jETI/jABC: Verification and Export. In *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Tech-*

nology, WI-IAT Workshop, pages 144–147, Silicon Valley, California, USA, November 2007. IEEE Computer Society Press.

[Kop14] Dawid Kopetzki. Model-based generation of graphical editors on the basis of abstract meta-model specifications. Master thesis, TU Dortmund, June 2014.

[KPJ98] Uwe Kastens, Peter Pfahler, and Matthias T. Jung. The Eli System. In *Proc. of the 7th Int. Conf. on Compiler Construction (CC '98)*, volume 1383 of *LNCS*, pages 294–297. Springer, 1998.

[KRbA+10] Dimitrios S. Kolovos, Louis M. Rose, Saad bin Abid, Richard F. Paige, Fiona A. C. Polack, and Goetz Botterweck. Taming EMF and GMF Using Model Transformation. In *Proc. of the 13th Int. Conf. on Model Driven Engineering Languages and Systems (MODELS 2010)*, pages 211–225, 2010.

[KRGDP15] Dimitris Kolovos, Louis Rose, Antonio García-Domínguez, and Richard Paige. *The Epsilon Book*. Published online: http://eclipse.org/epsilon/doc/book/, 2015. Last update: February 4, 2015.

[KT08] Steven Kelly and Juha-Pekka Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Press, Hoboken, NJ, USA, 2008.

[Lam13] Anna-Lena Lamprecht. *User-Level Workflow Design - A Bioinformatics Perspective*, volume 8311 of *Lecture Notes in Computer Science*. Springer, 2013.

[LKR14] Kevin Lano and Shekoufeh Kolahdouz-Rahimi. Model-Transformation Design Patterns. *IEEE Transactions on Software Engineering (TSE)*, 40(12):1224–1259, 12 2014.

[LMB+01] Akos Ledeczi, Miklos Maroti, Arpad Bakay, Gabor Karsai, Jason Garrett, Chuck Thomasson, Greg Nordstrom, Jonathan Sprinkle, and Peter Volgyesi. The Generic Modeling Environment. In *Workshop on Intelligent Signal Processing (WISP 2001)*, 2001.

[LMV03] Akos Lédeczi, Miklós Maróti, and Péter Völgyesi. The Generic Modeling Environment. Technical report, Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, 37221, USA, 2003.

[LNMS10] Anna-Lena Lamprecht, Stefan Naujokat, Tiziana Margaria, and Bernhard Steffen. Synthesis-Based Loose Programming. In *Proc. of the 7th Int. Conf. on the Quality of Information and Communications Technology (QUATIC 2010), Porto, Portugal*, pages 262–267. IEEE, September 2010.

[Lyb12] Michael Lybecait. Entwicklung und Implementierung eines Frameworks zur grafischen Modellierung von Modelltransformationen auf Basis von EMF-Metamodellen und Genesys. diploma thesis, TU Dortmund, 2012.

[MB02] Stephen J. Mellor and Marc J. Balcer. *Executable UML: A Foundation for Model-Driven Architecture*. Addison-Wesley Professional, 2002.

[MKS08] Tiziana Margaria, Christian Kubczak, and Bernhard Steffen. Bio-jETI: a service integration, design, and provisioning platform for orchestrated bioinformatics processes. *BMC Bioinformatics*, 9 Suppl 4:S12, 2008.

[MLA10] Jeff McAffer, Jean-Michel Lemieux, and Chris Aniszczyk. *Eclipse Rich Client Platform*. Addison-Wesley Professional, 2nd edition, 2010.

[MNS05] Tiziana Margaria, Ralf Nagel, and Bernhard Steffen. jETI: A Tool for Remote Tool Integration. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 3440/2005 of *LNCS*, page 557–562. Springer Berlin/Heidelberg, 2005.

[MS04] Tiziana Margaria and Bernhard Steffen. Lightweight coarse-grained coordination: a scalable system-level approach. *Software Tools for Technology Transfer*, 5(2-3):107–123, 2004.

[MS09] Tiziana Margaria and Bernhard Steffen. Business Process Modelling in the jABC: The One-Thing-Approach. In Jorge Cardoso and Wil van der Aalst, editors, *Handbook of Research on Business Process Modeling*. IGI Global, 2009.

[MSR05] Tiziana Margaria, Bernhard Steffen, and Manfred Reitenspieß. Service-Oriented Design: The Roots. In *Proc. of the 3rd Int. Conf. on Service-Oriented Computing (ICSOC 2005), Amsterdam, The Netherlands*, volume 3826 of *LNCS*, pages 450–464. Springer, 2005.

[MSR06] Tiziana Margaria, Bernhard Steffen, and Manfred Reitenspieš. Service-Oriented Design: The jABC Approach. In Francisco Cubera, Bernd J. Krämer, and Michael P. Papazoglou, editors, *Service Oriented Computing (SOC)*, number 05462 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[NLKS17] Stefan Naujokat, Michael Lybecait, Dawid Kopetzki, and Bernhard Steffen. CINCO: A Simplicity-Driven Approach to Full Generation of Domain-Specific Graphical Modeling Tools. *Software Tools for Technology Transfer*, 2017.

[NLS11] Stefan Naujokat, Anna-Lena Lamprecht, and Bernhard Steffen. Tailoring Process Synthesis to Domain Characteristics. In *Proc. of the 16th IEEE Int. Conf. on Engineering of Complex Computer Systems (ICECCS 2011), Las Vegas, NV, USA*, pages 167–175, 2011.

[NLS12a] Stefan Naujokat, Anna-Lena Lamprecht, and Bernhard Steffen. Loose Programming with PROPHETS. In Juan de Lara and Andrea Zisman, editors, *Proc. of the 15th Int. Conf. on Fundamental Approaches to Software Engineering (FASE 2012), Tallinn, Estonia*, volume 7212 of *LNCS*, pages 94–98. Springer Heidelberg, 2012.

[NLS$^+$12b] Stefan Naujokat, Anna-Lena Lamprecht, Bernhard Steffen, Sven Jörges, and Tiziana Margaria. Simplicity Principles for Plug-In Development: The jABC Approach. In *Proceedings of the Second International Workshop on Developing Tools as Plug-Ins (TOPI 2012), Zürich, Switzerland*, pages 7–12, 2012.

[NNL$^+$13] Stefan Naujokat, Johannes Neubauer, Anna-Lena Lamprecht, Bernhard Steffen, Sven Jörges, and Tiziana Margaria. Simplicity-First Model-Based Plug-In Development. *Software: Practice and Experience*, 44(3):277–297, December 2013. first published online.

[NNMS16] Stefan Naujokat, Johannes Neubauer, Tiziana Margaria, and Bernhard Steffen. Meta-Level Reuse for Mastering Domain Specialization. In *Proc. of the 7th Int. Symp. on Leveraging Applications of Formal Methods, Verification and Validation, Part II (ISoLA 2016)*, volume 9953 of *LNCS*, pages 218–237. Springer, 2016.

[NS13a] Johannes Neubauer and Bernhard Steffen. Plug-and-Play Higher-Order Process Integration. *IEEE Computer*, 46(11):56–62, August 2013.

[NS13b] Johannes Neubauer and Bernhard Steffen. Second-Order Servification. In Georg Herzwurm and Tiziana Margaria, editors, *Software Business. From Physical Products to Software Services and Solutions*, volume 150 of *Lecture Notes in Business Information Processing*, pages 13–25. Springer Berlin Heidelberg, 2013.

[NSM13] Johannes Neubauer, Bernhard Steffen, and Tiziana Margaria. Higher-Order Process Modeling: Product-Lining, Variability Modeling and Beyond. *Electronic Proceedings in Theoretical Computer Science*, 129:259–283, 2013.

[NTI+14] Stefan Naujokat, Louis-Marie Traonouez, Malte Isberner, Bernhard Steffen, and Axel Legay. Domain-Specific Code Generator Modeling: A Case Study for Multi-faceted Concurrent Systems. In *Proc. of the 6th Int. Symp. on Leveraging Applications of Formal Methods, Verification and Validation, Part I (ISoLA 2014)*, volume 8802 of *LNCS*, pages 463–480. Springer, 2014.

[Pet66] Carl Adam Petri. *Communication with automata*. PhD thesis, Universität Hamburg, 1966.

[Pet13] Marian Petre. UML in practice. In *Proc. of the 2013 Int. Conf. on Software Engineering (ICSE 2013)*, pages 722–731. IEEE Press Piscataway, NJ, USA, 2013.

[RCL09] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. A Taxonomy and Survey of Cloud Computing Systems. In *Proc. of 5th Int. Joint Conf. on INC, IMS and IDC, 2009. NCM '09*. IEEE, 2009.

[Rei85] Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *Monographs in Theoretical Computer Science. An EATCS Series*. Springer, 1985.

[RJB04] James Rumbaugh, Ivar Jacobsen, and Grady Booch. *The Unified Modeling Language Reference Manual*. The Addison-Wesley Object Technology Series. Addison-Wesley Professional, 2 edition, 7 2004.

[SBPM08] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework (2nd Edition)*. Addison-Wesley, Boston, MA, USA, 2008.

[SCK08] Carsten Schmidt, Bastian Cramer, and Uwe Kastens. Generating visual structure editors from high-level specifications. Technical report, University of Paderborn, Germany, 2008.

[SM99] Bernhard Steffen and Tiziana Margaria. METAFrame in Practice: Design of Intelligent Network Services. In Ernst-Rüdiger Olderog and Bernhard Steffen, editors, *Correct System Design*, volume 1710 of *Lecture Notes in Computer Science*, pages 390–415. Springer, 1999.

[SMB97] Bernhard Steffen, Tiziana Margaria, and Volker Braun. The Electronic Tool Integration platform: concepts and design. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1-2):9–30, 1997.

[SMBK97] Bernhard Steffen, Tiziana Margaria, Volkar Braun, and Nina Kalt. Hierarchical Service Definition. *Annual Review of Communications of the ACM*, 51:847–856, 1997.

[SMCB96] Bernhard Steffen, Tiziana Margaria, Andreas Claßen, and Volker Braun. The METAFrame'95 Environment. In *CAV*, pages 450–453, 1996.

[SMN+07] Bernhard Steffen, Tiziana Margaria, Ralf Nagel, Sven Jörges, and Christian Kubczak. Model-Driven Development with the jABC. In Eyal Bin, Avi Ziv, and Shmuel Ur, editors, *Hardware and Software, Verification and Testing*, volume 4383 of *Lecture Notes in Computer Science*, pages 92–108. Springer Berlin / Heidelberg, 2007.

[SN16] Bernhard Steffen and Stefan Naujokat. Archimedean Points: The Essence for Mastering Change. *LNCS Transactions on Foundations for Mastering Change (FoMaC)*, 1(1):22–46, 2016.

[STK+16] Alexander Schäferdiek, Benjamin Tokgöz, Fabian Kotschenreuter, Hang Yu, Jan Möller, Konstantin Tkachuk, Patrik Elfer, Ruikun Chang, and Volkan Gümüs. PG 592 - Planspiele für Krisenmanagement in Rechenzentren. Technical report, TU Dortmund, 2016.

[Tam14] Laurette Mariella Tamdjokwen. Grafische Modellierung der Auslegung von Pasteurisiermaschinen. Master's thesis, TU Dortmund, 2014. Bachelor thesis.

[Vli98] John Vlissides. *Pattern Hatching: Design Patterns Applied*. Addison-Wesley Professional, 1998.

[Wec16] Jan Weckwerth. Cinco Evaluation: CMMN-Modellierung und -Ausführung in der Praxis. Master's thesis, TU Dortmund, 2016.

[Wei67] Clark Weissman. *LISP 1.5 Primer*. Dickenson Publishing Company, Inc., Belmont, CA, USA, 1967.

[Wir15] Dominic Wirkner. Merge-Strategien für Graphmodelle am Beispiel von jABC und Git. Diploma thesis, TU Dortmund, February 2015.

[WMN16] Nils Wortmann, Malte Michel, and Stefan Naujokat. A Fully Model-Based Approach to Software Development for Industrial Centrifuges. In *Proc. of the 7th Int. Symp. on Leveraging Applications of Formal Methods, Verification and Validation, Part II (ISoLA 2016)*, volume 9953 of *LNCS*, pages 774–783. Springer, 2016.

[Wor15] Nils Wortmann. Modellbasierte Modellierung von industriellen Zentrifugen mit Codegenerierung für Steuerungssysteme. Bachelor thesis, Münster University of Applied Sciences, 2015.

[ZGH04] Nianping Zhu, John Grundy, and John Hosking. Pounamu: A Meta-Tool for Multi-View Visual Language Environment Construction. In *2004 IEEE Symposium on Visual Languages and Human Centric Computing*, 2004.

[Zwe15] Philip Zweihoff. Cinco Products for the Web. Master thesis, TU Dortmund, November 2015.

Online References

[1] Definition of Service by Merriam-Webster. http://www.merriam-webster.com/dictionary/service. [Online; last accessed 27-November-2016].

[2] Documents associated with Meta Object Facility^{TM} (MOF^{TM}) Version 2.5.1. http://www.omg.org/spec/MOF/2.5.1/. [Online; last accessed 29-November-2016].

[3] Eclipse CDT (C/C++ Development Tooling). http://www.eclipse.org/cdt/. [Online; last accessed 16-December-2016].

[4] Eclipse Java Development Tools (JDT). http://www.eclipse.org/jdt/. [Online; last accessed 16-December-2016].

[5] Eclipse Modeling Project. http://www.eclipse.org/modeling/. [Online; last accessed 10-December-2016].

[6] Eclipse Sirius. http://www.eclipse.org/sirius/. [Online; last accessed 06-November-2016].

[7] Epsilon. http://www.eclipse.org/epsilon/. [Online; last accessed 06-November-2016].

[8] Epsilon EuGENia. http://www.eclipse.org/epsilon/doc/eugenia/. [Online; last accessed 06-November-2016].

[9] Graphical Modeling Framework (GMF) Tooling. http://eclipse.org/gmf-tooling/. [Online; last accessed 10-December-2016].

[10] Graphiti - a Graphical Tooling Infrastructure. http://www.eclipse.org/graphiti/. [Online; last accessed 06-November-2016].

[11] Jasmin Home Page. http://jasmin.sourceforge.net/. [Online; last accessed 22-November-2016].

[12] Kotlin Programming Language. http://kotlinlang.org/. [Online; last accessed 23-November-2016].

[13] Marama. https://wiki.auckland.ac.nz/display/csidst/Welcome. [Online; last accessed 06-November-2016].

[14] MetaCase - Domain-Specific Modeling with MetaEdit+. http://www.metacase.com. [Online; last accessed 06-November-2016].

[15] Ruby Programming Language. http://www.ruby-lang.org/. [Online; last accessed 02-December-2016].

[16] Spray - a quick way of creating Graphiti. http://code.google.com/a/eclipselabs.org/p/spray/.

[Online; last accessed 06-November-2016].

[17] Swift.org. https://swift.org/. [Online; last accessed 23-November-2016].

[18] The Groovy programming language. http://groovy-lang.org/. [Online; last accessed 23-November-2016].

[19] The Scala Programming Language. http://www.scala-lang.org. [Online; last accessed 22-November-2016].

[20] WebGME. https://webgme.org/. [Online; last accessed 02-November-2016].

[21] Xtend - Modernized Java. http://xtend-lang.org. [Online; last accessed 03-November-2016].

[22] Xtext - Language Engineering Made Easy! http://www.eclipse.org/Xtext/. [Online; last accessed 06-November-2016].

[23] Apache Software Foundation. The Apache Velocity Project. https://velocity.apache.org/. [Online; last accessed 16-December-2016].

[24] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web Services Architecture. http://www.w3.org/TR/ws-arch/, 2 2004. W3C Working Group Note [Online; last accessed 02-December-2016].

[25] Jordi Cabot. Clarifying concepts: MBE vs MDE vs MDD vs MDA. http://modeling-languages.com/clarifying-concepts-mbe-vs-mde-vs-mdd-vs-mda/. [Online; last accessed 21-November-2016].

[26] Martin Fowler. Language Workbench. http://martinfowler.com/bliki/LanguageWorkbench.html. [Online; last accessed 28-October-2016].

[27] Martin Fowler. PostIntelliJ. http://martinfowler.com/bliki/PostIntelliJ.html. [Online; last accessed 29-November-2016].

[28] Martin Fowler. Fluent Interface. http://martinfowler.com/bliki/FluentInterface.html, 12 2005. [Online; last accessed 28-October-2016].

[29] Martin Fowler. Language Workbenches: The Killer-App for Domain Specific Languages? http://martinfowler.com/articles/languageWorkbench.html, 06 2005. [Online; last accessed 28-October-2016].

[30] JetBrains. IntelliJ IDEA. https://www.jetbrains.com/idea/. [Online; last accessed 29-November-2016].

[31] Anne Thomas Manes. SOA is Dead; Long Live Services. http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html, 01 2009. [Online; last accessed 25-November-2016].

[32] Joe McKendrick. Four reasons to embrace service-oriented thinking in 2011. http://www.zdnet.com/article/four-reasons-to-embrace-service-oriented-thinking-in-2011/, 12 2010. [Online; last accessed 28-November-2016].

[33] Dhananjay Nene. SOA ain't dead but it certainly is transforming. http://blog.dhananjaynene.com/2009/01/soa-aint-dead-but-it-certainly-is-transforming/, 01 2009. [Online; last accessed 27-November-2016].

[34] Object Management Group (OMG). UML Profile for BPMN Processes. http://www.omg.org/spec/BPMNProfile/. [Online; last accessed 23-November-2016].

[35] Object Management Group (OMG). Documents associated with UML®, Version 1.3. http://www.omg.org/spec/UML/1.3/, 09 2001. [Online; last accessed 29-November-2016].

[36] Object Management Group (OMG). Documents Associated with BPMN Version 2.0.1. http://www.omg.org/spec/BPMN/2.0.1/, 9 2013. [Online; last accessed 02-December-2016].

[37] Object Management Group (OMG). Documents Associated with Concrete Syntax for a UML Action Language: Action Language for Foundational UML^{TM} (ALF^{TM}), version 1.0.1. http://www.omg.org/spec/ALF/1.0.1/, 09 2013. [Online; last accessed 29-November-2016].

[38] Object Management Group (OMG). Documents Associated with Case Management Model and Notation (CMMN), Version 1.0. http://www.omg.org/spec/CMMN/1.0/, 5 2014. [Online; last accessed 25-October-2016].

[39] Object Management Group (OMG). Documents associated with Object Constraint Language (OCL), Version 2.4. http://www.omg.org/spec/OCL/2.4/, 2 2014. [Online; last accessed 02-December-2016].

[40] Object Management Group (OMG). Documents Associated with Decision Model and Notation (DMN), Version 1.1. http://www.omg.org/spec/DMN/1.1/, 06 2016. [Online; last accessed 29-November-2016].

[41] Object Management Group (OMG). Documents Associated with Semantics of a Foundational Subset for Executable UML Models (FUML^{TM}), v1.2.1. http://www.omg.org/spec/FUML/1.2.1/, 01 2016. [Online; last accessed 29-November-2016].

[42] Terence Parr. StringTemplate. http://www.stringtemplate.org/. [Online; last accessed 16-December-2016].

[43] Nico Rehwaldt. Element Templates in the Camunda Modeler | Camunda Team Blog. https://blog.camunda.org/post/2016/05/camunda-modeler-element-templates/. [Online; last accessed 23-November-2016].

[44] Markus Völter. MD*/DSL Best Practices - Version 2.0. http://voelter.de/data/pub/DSLBestPractices-2011Update.pdf, 03 2011. [Online; last accessed 21-November-2016].