Chapter 10
Service-Orientation: Conquering Complexity with XMDD

Tiziana Margaria and Bernhard Steffen

10.1 Motivation

Industrial practice of application development and integration is increasingly characterized by vaguely defined but urgent IT needs. Following pressure, external (by the market or by changed regulations), or internal (by a merger, or for improvement), it is clear that things (be they products, applications, or the own IT landscape) must be changed, but how? Answering this question is typically impossible before major parts of a realization are in place. This is due to the fact that only concrete artifacts provide a sufficiently stable ground for a common understanding between the involved stakeholders. Moreover, only when the customer has a tangible understanding of the options, can he effectively criticize and decide. One observes over and over again that in today's practice this kind of criticism starts only after a first release of a system, and that it continues along the whole life cycle. This observation makes *agility a* if not *the* central requirement for industrial system design. The problem here is twofold: how to manage the time pressure with adequate early feedback to the process owners, and how to manage the evolution of the systems over a long and very heterogeneous lifetime, where further integration, repurposing, and retargeting continuously changes the requirements on the fly. To do this, we need a sort of agile and lean form of "complexity engineering" that should ideally be intrinsic in the development method and help align the needs of the business customers (the process owners—who know the '*what*') with the resulting implementation (the 'how') [31].

10.1.1 Complexity Engineering

Complexity of systems comes in very different flavors and dimension, e.g.,

- sheer size—of the solution itself or of the entities to be processed;

T. Margaria (✉)
Chair Service and Software Engineering, University of Potsdam, Potsdam, Germany
e-mail: margaria@cs.uni-potsdam.de

M. Hinchey, L. Coyle (eds.), *Conquering Complexity*,
DOI 10.1007/978-1-4471-2297-5_10,

- conceptual complexity—the difficulty to understand any potential solution;
- and heterogeneity—the problem of integrating numerous partners, (communication) technologies, tools, and devices,

all of which can be again individually distinguished in *inherent* and *actual* complexity. Here, the inherent complexity is due to the tackled problem, and cannot be reduced without changing the problem, whereas the actual complexity refers to the complexity of an actual solution, which often is much higher than the inherent complexity.

The actually *felt* complexity may still be quite a different matter: 'Divide and Conquer', or the 'Separation of Concerns' may split a global complexity into a number of aspect-specific complexities, each of which, individually, may well be comfortably tackled at different times, by different people, with different means, often exploiting powerful standard solutions. For example, designing a complex business process can be done independently of integrating the involved applications and devices, independently of managing the often thousands of corresponding process instances on a network, and in particular, independently of the construction of the Internet, without which worldwide end-to-end processes would be hardly possible.

In this chapter we want to address the importance and the role of the 'felt' complexity/ies, which of course is quite subjective: the 'felt' complexity of any system depends on the individual roles sensing it. Usually, the business process designer does not feel the complexity of the realization and of the enactment, let alone the complexity of the required infrastructure. Conversely, those responsible for the infrastructure may not feel the complexity of the business-critical End-to-End process, its legal and economic consequences, and its vital implications for the company.

The central issue for a good and informed design of complex applications is therefore a method that reconciles the subjective views and competencies of the individual stakeholders into an adequate joint communication and decision-making framework. The goal is to comfortably manage an adequate division of labor and allow to easily exploit standards and available solutions in order to minimize the felt complexity for all stakeholders.

10.1.2 Extreme Model-Driven Development

Extreme Model-Driven Development (XMDD) combines into a coherent paradigm the decisive traits taken from:

- *eXtreme programming*, for providing immediate feedback through requirement and design validation by means of model tracing, simulation, and early testing;
- *service orientation*, for virtualizing the implementation of functionality;
- *aspect orientation*, for treating crosscutting as well as role specific concerns modularly; and
- *model-driven design*, for controlling the overall development at the modeling level.

Of course, XMDD cannot reduce the inherent complexity, but it can help make it explicit and thus improve its understanding and its management. Indeed, XMDD might probably add to the inherent complexity, but with the result that the individually felt complexities are rather low, due to the leveraging of standards, the division of labor, and due to the "80/20" principle, i.e. the approach where the majority of problems can be tackled easily, resorting to standard solutions, while only the few really specific and tough problems are left for special consideration.

Consider Graphical User Interface (GUI) design: 10–20 years ago building a GUI was a major project, involving substantial programming effort and pioneering creativity—it often involved PhD level work. With today's GUI libraries even beginners can produce quite advanced standard GUIs in a matter of hours. The inherent complexity of GUI design has not changed, but the advances in the foundational technologies and standards make today the development of a standard GUI rather easy. Another example is the development of parsers or compilers: in the 1970s, writing a compiler was an art. Today most parsers and compilers are easily generated. Thus an originally major problem turned into commodity without the inherent complexity being changed. To master unchanged and even increasing inherent complexity of a system in such a way that the felt complexity of the solution is understandable, acceptable, and manageable by the different stakeholders is a matter of adequate management of the actually felt complexity: this is the design space that we need to be able to explore and adapt to during the creation of a new application and throughout its lifetime. This is a question of agility and evolution.

10.1.3 Agility and Evolution

Separation of concerns and the lowering of the actually felt complexity are particularly important when agility is required and the solution must be able to react flexibly and quickly to new requirements and changed frame conditions. This agility is particularly necessary in areas like business processes, where the ability to change may be business critical. In the aftermath of the 9/11 terrorist attacks in 2001, airlines suffered immense losses because they were unable to adapt their business processes quickly enough to the changed market conditions and demand, ending in huge operational losses and insolvencies. Of course, situations like this are special, but they happen more often than one realizes: after 9/11 the same industry suffered similar crises in the aftermath of the SARS outbreak in 2006, and the Eyjafjallajokull volcanic eruption in 2010. So, there is no doubt that business processes are under the continuous pressure of change and in demand of powerful methods to manage the corresponding process evolution.

We will therefore focus on process modeling, agility, and evolution: How can large End-to-End processes be seamlessly and immediately adapted to new needs? Here is where ideas from eXtreme Programming (XP) enter the picture, and where our One Thing Approach (OTA) has its major impact. We will show how we achieve (a) application-level control, i.e., the continuous involvement of the customer and

application/business expert along the entire systems' life cycle, including software maintenance and evolution, together with (b) continuous and ongoing quality assurance with different means at different levels and phases (requirement validation, simulation, model checking, data flow analysis, testing, and monitoring), and (c) specific support to easily and non-invasively integrate new technologies, in a service-oriented way.

The key to our approach is to view the whole development process simply as a complex hierarchical and interactive decision process, where each stakeholder, including the application expert, is allowed to continuously place his/her decisions in term of constraints, and each development or evolution step can be regarded simply as a *transformation* of this set of constraints. We use constraints to describe all the pieces of knowledge and information that define and thus restrict the set of behaviors of the system. They comprise temporal constraints, loose process models, symbolic typing, as well as the definition of roles and rights. This allows one to continuously and globally monitor the consistency of the development and of the evolution process via varying forms of constraint checking.

In the remainder of the chapter we will elaborate on these ideas in more detail. Section 10.2 points to the importance of compatibility and interoperability as central meta-constraints of any system, then Sect. 10.3 summarizes the XMDD approach. Sections 10.4, 10.5 and 10.6 sketch our technical solution. Finally we will present some case studies in Sect. 10.7, before we conclude in Sect. 10.8.

10.2 Technical Hurdles: Compatibility and Interoperability

Today's systems require an unacceptable effort for deployment, which is typically caused by incompatibilities, feature interactions, and the sometimes catastrophic behavior of component upgrades, which no longer behave as expected. This gets even worse when considering heterogeneous, cross-organizational systems, whose components and interfaces typically evolve independently. Thus it is almost impossible to keep up with the increasing pace of changing market requirements.

This situation arises mainly due to the level on which systems are technically composed: even though high level languages and even model-driven development are used for component development, the system-level point of view is not yet adequately supported. In particular, the deployment of a heterogeneous systems is still a matter of assembly-level search for the reasons of incompatibility, which may be due to minimal version changes, slight hardware incompatibilities, or simply due to bugs, which come to surface only in a new, collaborative context of application. Integration testing and the quest for 'true' interoperability are major cost factors and major risks during a system implementation and deployment.

The hardware industry faced similar problems with even more dramatic consequences a decade ago: hardware is by nature far more difficult to patch, making failure of compatibility a real disaster. The trend since the late 1990s has been to move beyond VLSI towards Systems-on-a-Chip in order to guarantee larger integration in both senses: physically, by compacting complex systems on a single chip

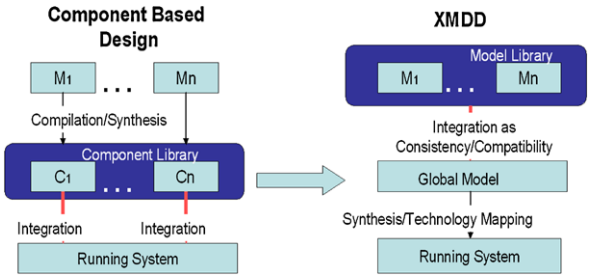

Fig. 10.1 The XMDD process

instead of physically wiring them on a board, but also conceptually, by integrating the components well before the silicon level, namely already at the design level. Rather than combining chips (the classical approach), hardware engineers started to directly combine the component's designs and to produce (in their terms, synthesize) system-level solutions that are homogeneous at the silicon level. Interestingly, they solve the problem of compatibility by moving it to a *higher level of abstraction* and going towards more homogeneous final products.

XMDD is a paradigm for application development that is conceptually closely related to the sketched SoC approach.

10.3 XMDD: Extreme Model-Driven Development

At the larger scale of system development, moving the problem of compatibility to a higher level of abstraction means moving it to the modeling level (see Fig. 10.1): rather than using the models, as is usual in the Component Based Development paradigm, just as a means of specification, which

- need to be compiled to become a 'real thing' (e.g., a component of a software library),
- must be updated (but typically are not), whenever the real thing changes, and
- typically only provide a local view of a portion or an aspect of a system,

models should be put at the center of the design activity, becoming *the* first class entities of the *global* system design process. In such an approach, as shown on the right side of Fig. 10.1,

- libraries should be established at the model level: building blocks should be (elementary) models rather than software components;
- systems should be specified by model combinations (composition, configuration, superposition, conjunction . . .), viewed as a set of constraints that the implementation needs to satisfy;
- global model combinations should be compiled (synthesized, e.g., by solving all the imposed constraints) into a homogeneous solution for a desired environment, which includes the realization of an adequate technology mapping;

- system changes (upgrades, customer-specific adaptations, new versions, etc.) should happen only (or at least primarily) at the model level, with a subsequent global recompilation (re-synthesis);
- optimizations should be kept distinct from design issues, in order to maintain the information on the structure and the design decisions independently of the considerations that lead to a particular optimized implementation.

Using XMDD—which strictly separates compatibility, migration, and optimization issues from model/functionality composition—it would be possible to overcome the problem of incompatibility between

- (global) models and (global) implementations, which is guaranteed and later-on maintained by (semi-)automatic compilation and synthesis, as well as between
- system components, paradigms, and hardware platforms: a dedicated compilation/synthesis of the considered *global* functionality for a specific platform architecture avoids the problems of incompatible design decisions for the individual components.

In essence, delaying the compilation/synthesis until all parameters are known (e.g., all compatibility constraints are available), may drastically simplify this task, as the individual parts can already be compiled/synthesized specifically for the current global context. In a good setup, this should not only simplify the integration issue (rather than having to be open for all eventualities, one can concentrate on precisely given circumstances), but also improve the efficiency of the compiled/synthesized implementations.

XMDD has the potential to drastically reduce the long-term costs due to version incompatibility, system migration and upgrading, and lower risk factors like vendor and technology dependency. Thus it helps protect investment in the software infrastructure. We are convinced that this extreme style of model-driven development will become the development style at least for mass-customized software in the future.

In particular we believe that XMDD, even though drastically different from state of the art industrial system design—which is itself driven right from the beginning by the underlying platform—will change the state of the art: technology moves so fast, and the varieties are so manifold that the classical platform-focused development will find its limits very soon.

10.4 Central Issues to be Addressed

In order to fully leverage the XMDD potential, and by this decrease the felt complexity, a number of issues need to be addressed:

- the design of adequate modeling patterns;
- the adaptation of analysis, verification, and compilation techniques and tools to the XMDD setting; and
- the realization of automatic deployment procedures.

10.4.1 Heterogeneous Landscape of Models

One of the major challenges for software engineering is that software is multi-dimensional: it comprises a number of different (loosely related) dimensions, which typically need to be modeled in different styles in order to be treated adequately. Important for simplifying the software/application development is the reduction of the complexity of this multi-dimensional space, by placing it into some standard scenario. Such reductions are typically application-specific. Besides simplifying the application development they also provide a handle for the required automatic compilation and deployment procedures.

Typical among these dimensions—also called *views*—are the following:

- The *(user) process view*, which describes the dynamic behavior of the system. How does it behave under each circumstance?
- The *architectural view*, which expresses the static structure of the software (dependencies like nesting, inheritance, and references). This should not be confused with the architectural view of the hardware platform, which may indeed be drastically different. The charm of the OO-style was that it claimed to bridge the gap to the user/process view.
- The *exception view*, which addresses the system's behavior under malicious or even unforeseen circumstances.
- The *timing view*, which captures real time aspects.
- The various *thematic views* concerned with roles, specific requirements, and other aspect-like points of view.

Of course, UML already tries to address all these facets in a unifying way. However, UML is currently rather a heterogeneous, expressive sample of languages, which lacks a clear notion of (conceptual) integration like consistency and the idea of global dynamic behavior. Such aspects are currently dealt with independently, e.g., by means of concepts like *contracts* [1] (or more generally, and more complexly, via business-rules oriented programming like e.g., in JRules.[1] The latter concepts are also not supported by systematic means for guaranteeing consistency. In contrast, XMDD views these heterogeneous specifications (consisting of essentially independent models) just as constraints which must be respected during the compilation/synthesis phase (see also [42]).

Another popular approach is Aspect Oriented Programming (AOP) [5, 13]. It has striking success stories for specific purposes (exception handling, access and timing control, insertion of assertions, etc.), but becomes rather intricate when used to solve more general problems. The idea here is to treat different aspects separately in the code, and then to weave the separate code fragments together. In general this requires a precise understanding of the weaving mechanism, which may be more complicated than programming the overall system traditionally. This is due to the fact that the claimed modularity is only in the file structure—not on the conceptual side. In other words, AOP allows one to write down the aspects separately,

[1] The JRules website is here: http://www.ilog.com/.

but understanding their mutual global impact may require a deep understanding of weaving, and, even worse, of the result of weaving, which very much reminds of an interleaving expansion of a highly distributed system.

10.4.2 Formal Methods and Tools

There are numerous formal methods and tools addressing validation, ranging from methods for correctness-by-construction/rule-based transformation, correctness calculi, model checkers, and constraint solvers to tools in practical use like PVS [41], Bandera [6], and SLAM [4] to name just a few. On the compiler side there are complex (optimizing) compiler suites, code generators, and controller synthesizers, and other methods to support technology mapping. A complete account of these methods is beyond the purpose of this chapter. Here it is sufficient to note that there is a high potential of available technology waiting to be used.

10.4.3 Automatic Deployment and Maintenance Support

This is the weakest point of the current practice: the deployment of complex systems on a heterogeneous, distributed platform is typically a nightmare, the required system-level testing is virtually unsupported, and maintenance and upgrading very often turn out to be extremely time consuming and expensive, de facto responsible for the slogan "never change a running system".

Still, in the same area there is a lot of technology one can build upon: the development of Java and the JVM or the .NET activities are well-accepted means to help getting models into operation, in particular, when heterogeneous hardware is concerned. Interoperability can be established using CORBA, RMI, RPC, Web services, complex middleware etc., and there are tools for testing and version management. Unfortunately, using these tools requires a lot of expertise, time to detect undocumented anomalies and to develop patches, and this for every application to be deployed.

XMDD differs radically from classical software development, which in our opinion is no longer adequate for the bulk of application programming, particularly when it comes to heterogeneous, cross-organizational systems which must adapt to rapidly changing market requirements. Accordingly, a new approach to system development needs to be developed.

10.5 The One Thing Approach

In XMDD, elaboration and refinement happen until a level is reached, where the classical requirement/implementation gap reduces to service-oriented realization

of user/application-level functionalities. Thus rather than building highly complex software architectures, XMDD is characterized by the management of complex hierarchical models that orchestrate/coordinate user/application-level functionalities.

This perspective is now solidified by the One Thing Approach (OTA), which combines the simplicity of the waterfall development paradigm with a maximum of agility [34]. Key to OTA is to view the entire development process simply as a cooperative hierarchical and interactive decision process, which is organized by building and refining one comprehensive model, the 'one thing'. Within this model, each stakeholder, including the application expert, is allowed to continuously place his/her decisions in term of constraints, and each development or evolution step can be regarded simply as a transformation of the current constraint set. These constraints, which may comprise all kinds of aspects, can e.g. be expressed in terms of

- (temporal) formulae expressing the intentions of the application, internal policies, legal constraints or technical frame conditions;
- (loose) process models, specifying the rough distributed workflow from the management perspective without concern for technicalities like type correctness, location or interoperability;
- (symbolic) type information, sufficient to imply executability (later to be enforced by our synthesis technology);
- definitions of roles and rights, timing and localization constraints, and exception handling, which are to be integrated during code generation in an aspect-oriented fashion.

In this view, the waterfall character of the development process is no longer a matter of development phase or a 'before/after', but rather a matter of the chosen decision hierarchy: who can decide/modify what, what is the binding power of which decisions, and how should conflicts be resolved. This approach, conceptually, allows one (1) to monitor globally the consistency of the development or evolution process simply via constraint checking, and (2) to impose a kind of decision hierarchy by mapping areas of competencies to roles of individuals, in order to identify required actions in case of constraint violation.

Like XP for programming in the small, this approach revolutionizes the process/application development process. It replaces the typically long (interaction-free) intervals between contract-and-requirements time and delivery-and-acceptance time, with all its pitfalls, with a continuous, cooperative development process. Misconceptions are revealed and can dealt with as they arise, and the understanding of the application under construction (the user experience) naturally builds up along the way. The new cooperative development and evolution style supports the agile adjustment by

- keeping the customer continuously up to date: the impact of each design decision on the application logic becomes apparent via the shared model, 'One Thing', which provides the customer with a continuously updated user experience;
- focusing on the application logic, which allows one to repair and modify right at the same level as where the need appears;

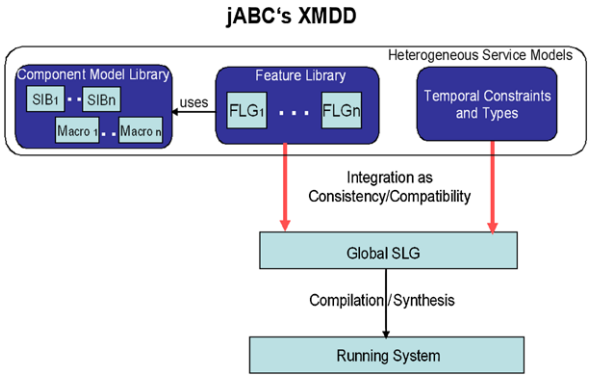

Fig. 10.2 The XMDD process in the jABC

- following the service-oriented paradigm making it is easy to exchange/integrate (third party) functionality in a non-invasive fashion.

10.6 The jABC as an XMDD Environment

The jABC [47, 48] is a framework designed to support systematic development according to the XMDD paradigm within the One Thing Approach. Developed at METAFrame Technologies in cooperation with the TU Dortmund, it is intended to promote the XMDD-style of development in order to move the responsibility and control of application development for certain classes of applications towards the application expert. In its current version[2] the jABC supports an agile and cooperative development of service-oriented systems along the lines of the One-Thing Approach. Technically it comprises the three features discussed above (cf. Fig. 10.2):

1. *Heterogeneous landscape of models*: the central model structure of the jABC are hierarchical Service Logic Graphs (SLGs) [30, 43]. SLGs are flowchart-like graphs. They model the application behavior in terms of the intended process flows, based on coarse granular building blocks called Service-Independent Building blocks (SIBs). These are intended to be understood directly by the application experts [43] i.e., independently of the structure of the underlying code, which in our case is typically written in Java/C/C++. The component models (SIBs or hierarchical subservices called GraphSIBs), the feature-based service models—called Feature Logic Graphs (FLGs)—and the Global SLGs modeling applications are all hierarchical SLGs.
   The jABC also supports model specification in terms of
   a. modal logics, to abstractly and loosely characterize valid behaviors: semantic linear time logic (SLTL) [32, 44] is used for synthesis and the branching time logic modal μ-calculus [14] for model checking,

[2] We refer to version 3.5 of jABC here.

b. a classification scheme for building blocks and types, and
c. high level type specifications, used to specify compatibility between the building blocks of the SLGs.

The granularity of the building blocks is essential here, as it determines the level of abstraction all the subsequent reasoning is based upon: the verification tools directly consider the SLGs as formal models, the names of the (parameterized) building blocks as (parameterized) events, and the branching conditions as (atomic) propositions. Thus the jABC focuses on the level of *component (SIB) composition* rather than on component construction: its compatibility, its type correctness, and its behavioral correctness are under formal methods' control [30].

2. *Formal methods and tools*: the jABC comprises a high-level type checker, two model checkers, a model synthesizer, a compiler for SLGs, an interpreter, and a view generator. The model synthesizer, the model checkers, and the type checker take care of the consistency and compatibility conditions expressed by the four kinds of constraints/models mentioned above.
3. *Automatic deployment and maintenance support*: an automated deployment process, system-level testing [39], regression testing, version control, and online monitoring [7] support the phases following initial deployment. In particular the automatic deployment service needs some meta-modeling in advance; that has been realized using the jABC itself. Likewise the testing services and the online monitoring are themselves strong formal methods-based [40] and have been realized via the jABC.

The jABC can be regarded as a first framework for XMDD. It is designed to continuously involve the customer/application expert throughout the whole systems' life cycle according to the OTA [34].

10.7 XMDD Case Studies in jABC

The XMDD paradigm has been successfully used in several contexts, at different abstraction levels. We will now illustrate how the jABC uniformly supports all the abstraction levels, from the requirements/design by non-IT experts in Sect. 10.7.1, to application design in Sect. 10.7.2, to middleware-level configurations in Sect. 10.7.3, complex, semantic web-enhanced processes in bioinformatics in Sect. 10.7.4, and the application to the construction of a family of re-targetable compilers in Sect. 10.7.5.

10.7.1 Requirements and Specification: Supply Chain Management

In [8] we concentrate on the collaborative design of complex embedded systems in the jABC, that has proven to be effective and adequate for team cooperation with

non-IT personnel. We show how our approach to model-driven collaborative design was applied to the requirement and specification phase of part of IKEA's P3 Document Management Process (part of a new Supply Chain Management system), where it complemented the Rational Unified Process development process already in use. The central contribution of our approach is two-dimensional support of consistency at the user process level:

- *vertical consistency* of models, e.g., across abstraction layers, as well as
- *horizontal model consistency*, which is needed, e.g. across organizational borders within a same abstraction level.

In this particular case we had to bridge between various business process specifications provided by business analysts on one side and use case/activity diagram views needed as specifications by the IT designers on the other side. Based on OTA, horizontal consistency was guaranteed by maintaining the global perspective throughout the refinement process, down to the code level, and vertical consistency by the simple discipline for refinement.

10.7.2 Application Construction: The SWS Challenge Mediation Scenario

A case study that demonstrates a wide span of XMDD features, from the design by modeling to the deployment and test, is our solution with jABC of the Mediation scenario of the Semantic Web Service (SWS) Challenge, as described in [16].

There, we show how we solved the Mediation task (a benchmark scenario of the Challenge, described in [23]) in a model driven, service oriented fashion using the jABC framework for model driven development and its jETI extension [44] for seamless integration of remote (Web) services. In particular we illustrate:

- how atomic services and orchestrations are modeled in the jABC;
- how legacy services and their proxies are represented within our framework, and how they are imported into our framework;
- how the mediators arise as orchestrations of the testbed's remote services and of local services;
- how vital properties of the Mediator are verified via model checking in the jABC; and
- how jABC/jETI orchestrated services are exported as Web services.

Besides providing a solution to the mediation problem, this also illustrates the *agility* of jABC-based solutions, since in the Challenge each scenario comprises a set of problems that come in different levels that build on top of each other. One of the central assessments is the ability of a methodology and of the corresponding technologies and tools to leverage on the first-level solutions to accommodate the changes/extensions required by the subsequent levels with minimal intrusion (in the solution and platforms) and effort (of a modeler/programmer).

The flexibility of the approach has been recently shown in two orthogonal directions:

- the flexibility of the automatic service composition via orchestration synthesis, which had been shown in [29, 32, 44] and demonstrated on the concrete case of the mediation scenario with a number of different construction principles, tools and algorithms in [15, 26];
- the flexibility in coping with changed platform realities—as is common in business evolution—that has been shown in [36] along two different directions of migration/extension of the underlying ERP platform.

10.7.3 Middleware Services: MaTRICS

In [3] we present how we realize in jABC the remote configuration and fault tolerance of the Online Conference Service [24] with our service oriented framework MaTRICS [2]. MaTRICS is our model-based service-oriented platform for remote intelligent configuration and management of systems and services. It is built on top of the jABC, thus it inherits the XMDD perspective. One of the central services offered by MaTRICS is the provision of low-overhead high-availability mechanisms for complex applications that run on distributed platforms. Our solution leaves the services untouched and uses the open source cluster management software, *heartbeat*[3] [38], to provide the high availability features. We showed there how jABC's XMDD approach supports the management services at, or close to, the middleware and operating system level, providing a user-friendly level of service models (implemented as SLGs according to the XMDD paradigm) for the monitoring (sensing of correct functionality) and the reconfiguration/service migration (actuating the changes on the cluster by steering heartbeat functionality). This is in contrast with the usual, script-based, heartbeat working manner, which is strictly code-based.

Reexamining the six issues mentioned in Sect. 10.1, in this case study:

- we structure the high-availability solution from the application perspective, for an application-level definition and management of the high-availability services well above the scripting level (user-centric modeling);
- we enable the model-level validation of the application logic (animation-based requirement validation and model checking), opposed to the sole testing possible in a script-based solution;
- we find an adequate, higher, and more declarative level, where application modeling is handed over to the implementation of (elementary) services. The library of services provided by MaTRICS has been extended by a new, reusable collection that internally uses heartbeat. This establishes a higher-level domain-specific language and service library for high-availability monitoring and enforcement;

[3]The Linux high-availability software website is here: http://www.linux-ha.org.

- we automatically deploy the new services, which are complex aggregations and enhanced compositions of the middleware services they embed;
- the new high-availability services and test cases are themselves monitorable at run time; and
- they are easily adaptable according to new requirements and to new platforms.

10.7.4 Bioinformatics Processes: Bio-JETI

Applying XMDD in the field of bioinformatics workflows led to the development of Bio-jETI [25] as a service platform for interdisciplinary work on biological application domains. The following advantages of the approach became evident for bioinformatics workflow management:

- *Integration of heterogeneous resources into a homogeneous environment*. With GeneFisher-P [20], for instance, we built a process-based variant of a software for Polymerase Chain Reaction (PCR) primer design.[4] Within GeneFisher-P we reuse several standard web services and a number of legacy tools that have been integrated with the help of the jETI technology [27] using the jABC modeling framework as the behavioral integration and interoperability layer. Within Bio-jETI, both these remote services and locally available auxiliary functionality (for tasks like file handling) have a uniform appearance (as SIBs) and can be used in the same fashion for workflow/process development.
- *Agility of workflow design*. With XMDD, even complex heterogeneous workflows can be easily changed or extended at the graphical level. In [17] we built several variations of workflows for the frequently needed multiple sequence alignment computation. We provided a set of preconfigured services and workflow snippets on a canvas, so that variations of an alignment workflow, for instance reading the input sequences either from a local file or from a remote database, or calling an alignment service either at the European Bioinformatics Institute or at the Bielefeld Bioinformatics Server, can be built by simply redirecting the branches between the services according to the intended workflow.
- *Deployment to different target platforms*. Using the Genesys code generation framework [9], Bio-jETI models can be compiled into different target languages. We translated a bioinformatics worfklow (performing a homology search and subsequently a multiple sequence alignment with the obtained sequences) into different flavors of native Java code and compared the execution times of the resulting applications, showing that the overhead that is introduced by the model-driven development process is negligible [18].

Moreover, the application of formal methods to support the development process is intended within OTA and has also become part of Bio-jETI: Model checking supports the detection of conceptual errors as well as of type inconsistencies,

[4] PCR primers are small nucleic acids that are required for initiating the amplification of DNA fragments.

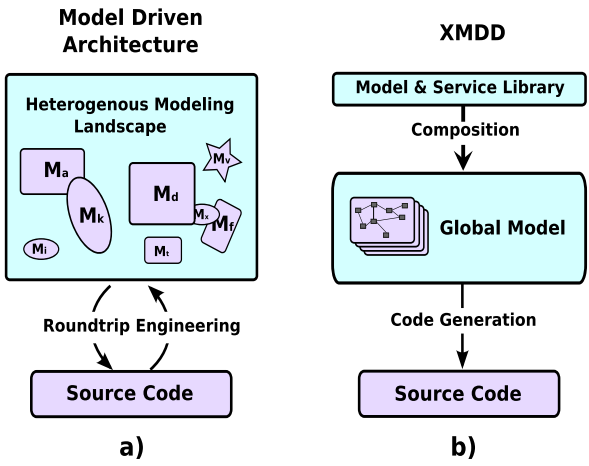

Fig. 10.3 From MDD to XMDD: no round trip engineering

whereas process synthesis methods can be applied to fill gaps within workflows automatically [19]. The bioinformatics community has made significant progress in equipping their services with metadata in terms of Semantic Web technology, and is thus often already providing the information that is needed for proper application of our synthesis techniques. For example, the European Molecular Biology Open Software Suite (EMBOSS) comprises around 350 biological sequence analysis tools, and the EMBRACE Ontology for Data and Methods (EDAM) ontology provides a controlled vocabulary for bioinformatics types and services. We showed in different case studies that (semi-)automatic workflow composition (of EMBOSS tools according to the EDAM ontology) delivers excellent results [18, 21] and can be exploited in practice within the XMDD concept, in what we call a *loose programming* approach [22].

10.7.5 Code Generation: The Genesys Framework

In contrast to the previous application areas, Genesys does not just profit from the XMDD approach, but is itself an important constituent of it: its fully automatic code generation capability allows the users of the jABC to design, control, and modify their process models at the application level, without any need for code-level modification, and consequently, without the burden of round-trip engineering [49] (see Fig. 10.3). Thanks to Genesys, generated code can be considered a "by-product" that must never be touched manually, as it can readily be obtained by full code generation.

Genesys [9, 10, 12] is a framework for the high-level engineering of code generators in XMDD fashion [30, 33, 35]: code generators are modeled as SLGs based on a model and service library that is specifically adapted for the domain of code generation. This library is constantly growing, as any newly developed artifact may immediately contribute to the library, which does not only comprise individual code

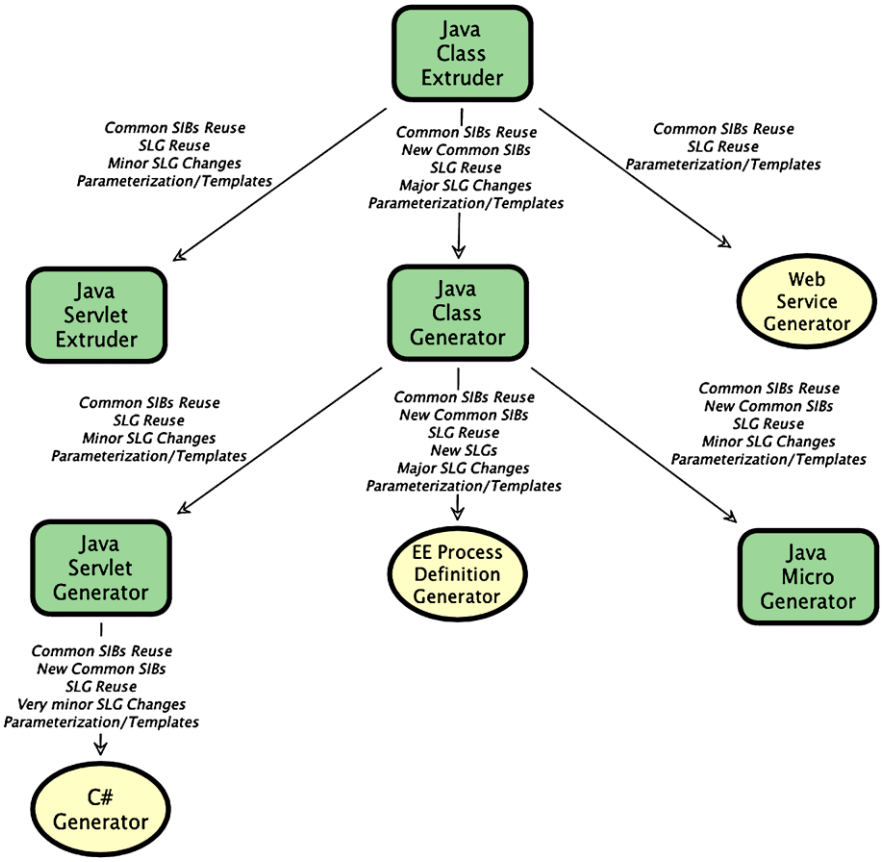

Fig. 10.4 Excerpt of the Genesys product line

generation functionalities, but also complex reusable features like error handling or code beautification, or even entire (models of) code generators. This is a consequence of the fact that SLGs can be truly hierarchical, which grants high reusability not only of the building blocks, but also of the models themselves, independent of their size [45].

Genesys also profits from jABC's clear support of service orientation [37], which allows one to seamlessly integrate third party functionality via SIBs [28]. This could be exploited to enhance Genesys' process-oriented modeling power with AndroMDA's[5] strength for modeling static aspects [11].

The close synergy between jABC and Genesys is best illustrated when looking at Genesys-generated code generators, which target Java, C#, Ruby, Objective-C, as well as BPEL, Lego Mindstorms, Android, iPhone, and more. Figure 10.4 shows an excerpt of this code generation 'product line', which can be extended and main-

[5]The AndroMDA website is here: http://www.andromda.org/.

tained within the jABC. This provides the means to validate the code generators at the model-level with respect to an increasing set of temporal properties expressing, e.g., the completeness of the applied tool chain or of the treatment of all involved artifacts, like parameters, local variables, and sub-models [9, 10, 46]. Experience shows that this approach significantly accelerates the development of new code generators by facilitating reuse of SIBs, models, and properties in a way that allows code generators to seamlessly evolve from each other [9, 35, 36].

10.8 Conclusions and Perspectives

We have advocated with XMMD a new direction for mastering of complexity in the service-oriented design of complex applications by combining ideas taken from eXtreme programming, model-driven design, as well as aspect and service orientation. Central is here the 'One-Thing Approach', which works by successively enriching and refining one single artifact, which, throughout the whole life-cycle, maintains a direct link between user-centric high-level models and the corresponding evolving running application. This approach is tailored to address the need for agile and lean development, which is particularly evident when it comes to heterogeneous, cross organizational systems which must adapt to rapidly changing market requirements. We have sketched the impact of this lightweight and cooperative development style that puts the *user process* at the center of development and the *application expert* in control of the process evolution by means of a number of case studies that indicate the breadth of applicability.

XMDD is not intended to replace genuine software development, as it assumes techniques to be able to solve problems (like synthesis or technology mapping) which are undecidable in general. On the other hand, more than 90% of the software development costs that arise worldwide concern a rather elementary software development level—as during routine application programming or software updates—where there are no technological or design challenges. There, the major problem faced is the management and control of software quantity, as it arises, e.g., through fast-evolving product lines, or instant solutions to solve an immediate but short-term need. XMDD is intended to address (a significant part of) this 90% 'niche'.

References

1. Andrade, L., Fiadeiro, J.L.: Architecture based evolution of software systems. In: Formal Methods for Software Architectures. Lecture Notes in Computer Science, vol. 2804, pp. 148–181. Springer, Berlin (2003)
2. Bajohr, M., Margaria, T.: MaTRICS: a service-based management tool for remote intelligent configuration of systems. Innovations Syst. Softw. Eng. **2**, 99–111 (2006)
3. Bajohr, M., Margaria, T.: High service availability in MaTRICS for the OCS. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Communications in Computer and Information Science, vol. 17, pp. 572–586. Springer, Berlin (2009)

4. Ball, T., Cook, B., Das, S., Rajamani, S.K.: Refining approximations in software predicate abstraction. In: Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29–April 2, 2004. Lecture Notes in Computer Science, vol. 2988, pp. 388–403. Springer, Berlin (2004)
5. Colyer, A., Clement, A., Harley, G., Webster, M.: Eclipse Aspectj: Aspect-Oriented Programming with Aspectj and the Eclipse Aspectj Development Tools. Addison-Wesley, Reading (2004)
6. Corbett, J.C., Dwyer, M.B., Hatcliff, J., Robbie: Bandera: a source-level interface for model checking Java programs. In: ICSE, pp. 762–765 (2000)
7. Hagerer, A., Hungar, H., Niese, O., Steffen, B.: Model generation by moderated regular extrapolation. In: Kutsche, R.-D., Weber, H. (eds.) Fundamental Approaches to Software Engineering, 5th International Conference, FASE 2002, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8–12, 2002. Lecture Notes in Computer Science, vol. 2306, pp. 80–95. Springer, Berlin (2002)
8. Hörmann, M., Margaria, T., Mender, T., Nagel, R., Steffen, B., Trinh, H.: The jABC approach to rigorous collaborative development of SCM applications. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation, Third International Symposium, ISoLA 2008, Porto Sani, Greece, October 13–15, 2008. Communications in Computer and Information Science, vol. 17, pp. 724–737. Springer, Berlin (2008)
9. Jörges, S., Margaria, T., Steffen, B.: Genesys: service-oriented construction of property conform code generators. Innovations Syst. Softw. Eng. **4**(4), 361–384 (2008)
10. Jörges, S., Margaria, T., Steffen, B.: Assuring property conformance of code generators via model checking. Form. Asp. Comput. 1–18 (2010). doi:10.1007/s00165-010-0169-9
11. Jörges, S., Steffen, B.: Leveraging service-orientation for combining code generation frameworks. In: 16th Annual IEEE International Conference on the Engineering of Complex Computer Systems (ICECCS), pp. 198–207 (2011)
12. Jörges, S., Steffen, B., Margaria, T.: Building code generators with Genesys: a tutorial introduction. In: 3rd International Summer School Conference on Generative and Transformational Techniques in Software Engineering III. GTTSE'09, pp. 364–385. Springer, Berlin (2011)
13. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, J.-M., Irwin, J.: Aspect-oriented programming. In: ECOOP, pp. 220–242 (1997)
14. Kozen, D.: Results on the propositional mu-calculus. Theor. Comput. Sci. **27**, 333–354 (1983)
15. Kubczak, C., Margaria, T., Steffen, B.: Mashup development for everybody: a planning-based approach. In: 3rd Int. Worksh. on Service Matchmaking and Resource Retrieval in the Semantic Web, Colocated with ISWC-2009, Washington, DC, USA. CEUR Workshop Proceedings, vol. 525 (2009)
16. Kubczak, C., Margaria, T., Steffen, B., Nagel, R.: Service-oriented mediation with jABC/jETI. In: Petrie, C., Margaria, T., Zaremba, M., Lausen, H. (eds.) Semantic Web Services Challenge: Results from the First Year, pp. 71–99. Springer, Berlin (2009)
17. Lamprecht, A.-L., Margaria, T., Steffen, B.: Seven variations of an alignment workflow—an illustration of agile process design and management in Bio-jETI. In: Mandoiu, I.I., Sunderraman, R., Zelikovsky, A. (eds.) Bioinformatics Research and Applications, Fourth International Symposium, ISBRA 2008, Atlanta, GA, USA, May 6–9, 2008. Lecture Notes in Computer Science, vol. 4983, pp. 445–456. Springer, Berlin (2008)
18. Lamprecht, A.-L., Margaria, T., Steffen, B.: Bio-jETI: a framework for semantics-based service composition. BMC Bioinform. **10**(S-10), 8 (2009)
19. Lamprecht, A.-L., Margaria, T., Steffen, B.: Supporting process development in Bio-jETI by model checking and synthesis. In: SWAT4LS-2009, Semantic Web Applications and Tools for Life Sciences. Proceedings of the Workshop on Semantic Web Applications and Tools for Life Sciences, Amsterdam, The Netherlands, November 20, 2009. CEUR Workshop Proceedings, vol. 559 (2009)
20. Lamprecht, A.-L., Margaria, T., Steffen, B., Sczyrba, A., Hartmeier, S., Giegerich, R.: GeneFisher-P: variations of genefisher as processes in Bio-jETI. BMC Bioinform. **9**(S-4) (2008). doi:10.1186/1471-2105-9-S4-S13

21. Lamprecht, A.-L., Naujokat, S., Margaria, T., Steffen, B.: Semantics-based composition of EMBOSS services. J. Biomed. Semant. **2**(Suppl 1), 5 (2011)
22. Lamprecht, A.-L., Naujokat, S., Steffen, B., Margaria, T.: Constraint-guided workflow composition based on the EDAM ontology. CoRR arXiv:1012.1640 (2010)
23. Lausen, H., Künster, U., Petrie, C., Zaremba, M., Komazec, S.: SWS challenge scenarios. In: Semantic Web Services Challenge Results from the First Year. Springer, Berlin (2009)
24. Margaria, T., Karusseit, M.: Community usage of the online conference service: an experience report from three CS conferences. In: Monteiro, J.L., Swatman, P.M.C., Tavares, L.V. (eds.) Towards the Knowledge Society: eCommerce, eBusiness, and eGovernment, the Second IFIP Conference on E-Commerce, E-Business, E-Government (I3E 2002), Lisbon, Portugal, October 7–9, 2002, pp. 497–511 (2002)
25. Margaria, T., Kubczak, C., Steffen, B.: Bio-jETI: a service integration, design, and provisioning platform for orchestrated bioinformatics processes. BMC Bioinform. **9**(S-4) (2008). doi:10.1186/1471-2105-9-S4-S12
26. Margaria, T., Meyer, D., Kubczak, C., Isberner, M., Steffen, B.: Synthesizing semantic web service compositions with jMosel and Golog. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25–29, 2009. Lecture Notes in Computer Science, vol. 5823, pp. 392–407. Springer, Berlin (2009)
27. Margaria, T., Nagel, R., Steffen, B.: jETI: a tool for remote tool integration. In: Halbwachs, N., Zuck, L.D. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, 4–8 April 2005, pp. 557–562 (2005)
28. Margaria, T., Nagel, R., Steffen, B.: Remote integration and coordination of verification tools in JETI. In: 12th IEEE International Conference on the Engineering of Computer-Based Systems (ECBS 2005), Greenbelt, MD, USA, 4–7 April 2005, pp. 431–436 (2005)
29. Margaria, T., Steffen, B.: Backtracking-free design planning by automatic synthesis in metaframe. In: FASE, pp. 188–204 (1998)
30. Margaria, T., Steffen, B.: Lightweight coarse-grained coordination: a scalable system-level approach. Int. J. Softw. Tools Technol. Transf. **5**(2–3), 107–123 (2004)
31. Margaria, T., Steffen, B.: From the how to the what. In: Meyer, B., Woodcock, J. (eds.) Verified Software: Theories, Tools, Experiments, First IFIP TC 2/WG 2.3 Conference, VSTTE 2005, Zurich, Switzerland, October 10–13, 2005, Revised Selected Papers and Discussions, pp. 448–459 (2005)
32. Margaria, T., Steffen, B.: LTL guided planning: revisiting automatic tool composition in ETI. In: 31st Annual IEEE / NASA Software Engineering Workshop (SEW-31 2007), Loyola College, Columbia, MD, USA, 6–8 March 2007, pp. 214–226 (2007)
33. Margaria, T., Steffen, B.: Agile IT: Thinking in user-centric models. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation, Third International Symposium, ISoLA 2008, Porto Sani, Greece, October 13–15, 2008, pp. 490–502 (2008)
34. Margaria, T., Steffen, B.: Business process modelling in the jABC: the one-thing approach. In: Cardoso, J., van der Aalst, W. (eds.) Handbook of Research on Business Process Modeling, pp. 1–26. IGI Global, Hershey (2009)
35. Margaria, T., Steffen, B.: Continuous model-driven engineering. IEEE Comput. **42**(10), 106–109 (2009)
36. Margaria, T., Steffen, B., Kubczak, C.: Evolution support in heterogeneous service-oriented landscapes. J. Braz. Comput. Soc. **16**(1), 35–47 (2010)
37. Margaria, T., Steffen, B., Reitenspieß, M.: Service-oriented design: the roots. In: Benatallah, B., Casati, F., Traverso, P. (eds.) Service-Oriented Computing—ICSOC 2005, Third International Conference, Amsterdam, The Netherlands, December 12–15, 2005. Lecture Notes in Computer Science, vol. 3826, pp. 450–464 (2005)
38. Marowsky-Brée, L.: A new cluster resource manager for heartbeat. In: UKUUG LISA/Winter Conf. on High-Availability and Reliability, Bournemouth (UK) (2004)

39. Niese, O., Margaria, T., Hagerer, A., Nagelmann, M., Steffen, B., Brune, G., Ide, H.-D.: An automated testing environment for CTI systems using concepts for specification and verification of workflows. Annu. Rev. Commun. **54**, 927–936 (2001)
40. Niese, O., Steffen, B., Margaria, T., Hagerer, A., Brune, G., Ide, H.-D.: Library-based design and consistency checking of system-level industrial test cases. In: Proceedings of the 4th International Conference on Fundamental Approaches to Software Engineering. FASE '01, pp. 233–248. Springer, London (2001)
41. Shankar, N., Owre, S.: Principles and pragmatics of subtyping in PVS. In: Bert, D., Choppy, C., Mosses, P.D. (eds.) Recent Trends in Algebraic Development Techniques, 14th International Workshop, WADT '99, Château de Bonas, France, September 15–18, 1999, Selected Papers, pp. 37–52 (1999)
42. Steffen, B.: Unifying models. In: Reischuk, R., Morvan, M. (eds.) STACS 97, 14th Annual Symposium on Theoretical Aspects of Computer Science, Lübeck, Germany, February 27–March 1, 1997, pp. 1–20 (1997)
43. Steffen, B., Margaria, T.: METAFrame in practice: design of intelligent network services. In: Olderog, E.-R., Steffen, B. (eds.) Correct System Design, Recent Insight and Advances, pp. 390–415, (to Hans Langmaack on the occasion of his retirement from his professorship at the University of Kiel) (1999)
44. Steffen, B., Margaria, T., Braun, V.: The electronic tool integration platform: concepts and design. Int. J. Softw. Tools Technol. Transf. **1**(1–2), 9–30 (1997)
45. Steffen, B., Margaria, T., Braun, V., Kalt, N.: Hierarchical service definition. Annu. Rev. Commun. **51**, 847–856 (1997)
46. Steffen, B., Margaria, T., Claßen, A., Braun, V.: Incremental formalization: a key to industrial success. Softw. Concepts Tools **17**(2), 78 (1996)
47. Steffen, B., Margaria, T., Nagel, R., Jörges, S., Kubczak, C.: Model-driven development with the jABC. In: Bin, E., Ziv, A., Ur, S. (eds.) Hardware and Software, Verification and Testing, Second International Haifa Verification Conference, HVC 2006, Haifa, Israel, October 23–26, 2006. Revised Selected Papers. Lecture Notes in Computer Science, vol. 4383, pp. 92–108. Springer, Berlin (2006)
48. Steffen, B., Narayan, P.: Full life-cycle support for end-to-end processes. IEEE Comput. **40**(11), 64–73 (2007)
49. Steffen, B., Wagner, C., Margaria, T.: Round-trip engineering vs. one-thing approach. In: Laplante, P.A. (ed.) Encyclopedia of Software Engineering. Auerbach Publications, Boca Raton (2010)