

A Model-Driven Approach to Business Software Engineering

Tobias HILDENBRAND

Department of Information Systems, University of Mannheim
68131 Mannheim, Baden-Wuerttemberg, Germany

and

Axel KORTHAUS

Department of Information Systems, University of Mannheim
68131 Mannheim, Baden-Wuerttemberg, Germany

ABSTRACT

Concurrent Engineering is a principle often implemented in product development processes in traditional engineering disciplines, e.g. the automotive industry. The advent of professional tools and standards supporting model-driven development (MDD) enables the creation of a similar scenario in the software industry. MDD tools can in a way be compared to Computer Aided Design (CAD) tools, and standardizing the output formats of such tools facilitates the comparison and consolidation of individual engineering artifacts in a parallel development process with one common architecture. In software engineering, performing frequently recurring consistency checks of the modeled architecture and individual components, respectively, can help prevent mismatches and inconsistencies in early stages of the software development cycle irrespective of the target platform, i.e. on a very high level of abstraction, when still not a single line of code has yet been written and adaptation is still cheap. This paper adumbrates a software development process suitable for creating business application software and domain-specific software assets. The process is embedded into a concurrent, collaborative and component-based methodology enabled by advanced MDD techniques.

Keywords: Model-Driven Development, Business Software, Concurrent Engineering, Collaborative Software Engineering, Component-Based Development, and Business Components

1. INTRODUCTION

Software development technologies, processes and related tool support have seen quite some innovations in the last ten to fifteen years. Aside from steadily raising the level of software development abstraction (from assembler via

modern programming languages like C and Java to partly graphical modeling languages like the UML), the concept of software components emerged, allowing for the definition of coarse-grained and self-contained software modules subsuming smaller entities, like classes and interfaces in object-oriented languages (component-based development, CBD). Those innovative approaches to raising the abstraction level in the development process and to increasing the granularity of reusable software development units can be the starting point for the definition of new, improved software development methodologies aiming at higher degrees of productivity through platform-independent, sustainable, and reusable software assets.

Currently, well-established general process frameworks like the *Unified Software Development Process* (UP) [13] or the German *V-Model*'97 as well as tailor-made processes like e.g. *Catalysis* [8] and *KobrA* [2] already provide support for certain up-to-date focal points of software development, i.e. architecture-centering or component-based product line engineering in the latter case, but they lack a more comprehensive combination of the most promising principles suitable for achieving enhanced productivity and quality, which we think are:

- reuse and
- collaboration.

The C³-Process for business application engineering adopts and elaborates on many principles from the Business Object Oriented Software Technology for Enterprise Reengineering (BOOSTER) approach [16], which in turn is based on best practices from the UP, *SELECT Perspective* [1], and the *Catalysis* Approach [8], among others. As BOOSTER constitutes the foundation for this paper, it is described in more detail in the following chapter, together with novel implications of MDD principles for the original BOOSTER approach and business component modeling in general. C³ comprises the concept of inter-organizational collaboration alongside a virtual “software supply chain”, which is backed up by a “persistence mechanism”

in the form of domain-specific repositories providing domain and component metadata. C³ also introduces concurrent software engineering techniques for both system architecture and component design realized by means of MDD and XML Metadata Interchange (XMI) [18]. As components represent the focal artifact in the proposed process model, the name C³ is composed of the three elements described above:

- Collaboration,
- Concurrent Software Engineering, and
- Component-Orientation.

2. MODELING BUSINESS COMPONENTS

Component-based development (CBD) aims at the modular composition of business applications by means of reusable, coarse-grained, self-contained, and marketable software building blocks.

Reuse is one key to enhanced software process productivity and consistent quality. According to Biggerstaff and Richter reusability techniques can be divided into two major categories [5]:

- composition technologies and
- generation technologies.

In order to increase software process productivity as a whole, we are intending to combine business software composition from business components and automatic code generation techniques from business component models, e.g. represented in a domain-specifically extended UML profile. That is, the proposed methodology tries to at least reduce the amount of platform-dependent, manually-coded artifacts produced in the process and reuse as coarse-grained software building blocks as possible.

Business Components

The concept of business components in enterprise application systems is a means of flexibly supporting business processes and exploiting reusable software assets [11]—either archived in-house or externally procured.

Fig. 1 shows the different levels of component specificity. Components on the enterprise level capture critical business logic of individual companies and are not likely to be shared in an open source environment. Domain-specific components, however, facilitate the establishment of a central domain repository archiving software assets. General business and system level components comprise standard functionality, e.g. customer management or components for printing etc. This classification of components was inspired by the Object Management Group’s (OMG) *Business Objects* specification [17].

BOOSTER [16] as a business component-based approach to application system development considers the whole software lifecycle. Thus, the BOOSTER methodology constitutes the foundation for the envisioned C³ software process, which is additionally integrates up-to-date as-

pects of model-driven component development and automated platform-specific code generation. The architecture of the current version of BOOSTER**Process* is depicted in Fig. 2.

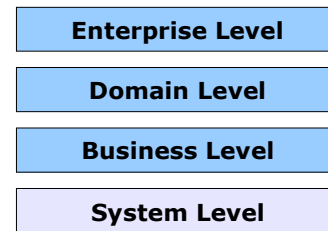


Fig. 1: Levels of Component Specificity

BOOSTER**Core*

In addition to the process architecture the BOOSTER approach also includes a technical conceptual framework consisting of a business component metamodel (cf. section 4) and the corresponding UML profile with specialized stereotypes, tagged values and constraints for business component modeling and specification. At the time of conceptualizing BOOSTER**Core*, OMG’s UML Profile for Enterprise Distributed Object Computing Specification (EDOC) [19] was not yet available. Now, this standardized and accepted UML profile substitutes some of the basic ideas presented in the original BOOSTER**Core* framework as introduced in [16].

Further extending the basic framework to semantically enrich the modeling capabilities to enable automatic code generation is the missing link to MDD within the BOOSTER approach.

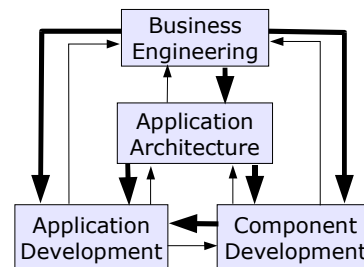


Fig. 2: BOOSTER**Process*

BOOSTER**Process*

In Fig. 2, the main activities in the BOOSTER**Process* are depicted from a macro perspective. The two different kinds of arrows indicate a more or less significant flow of information between those activities. In [16], the main activities are further elaborated on and refined by hierarchically subordinated activities. In this paper, we particularly focus on the elaboration of the application development and component development activities, and that, in turn, especially in terms of process control and project management aspects as illustrated in section 5.

3. COLLABORATIVE, CONCURRENT SOFTWARE DEVELOPMENT

Collaborative Software Development Approaches

Collaborative business processes, e.g. in supply chain management or enterprise resource planning contexts, become more and more the rule rather than the exception. Collaboration usually implies the involvement of at least two organizational units, either within or outside company boundaries. Current developments in the software industry also foster these principles in software development processes. The rising complexity of modern business application systems and ever shorter development cycles (time-to-market) create a very rough environment for individual software companies and development teams.

Collaborative software development (CSD)—often synonymously called global software development, distributed software engineering, or virtual software corporation—generally implies distributing software development activities over time, distance, or organizational borders [4]. In our approach, we mainly focus on remote, inter-organizational collaboration.

Concurrent Software Engineering

Already well known from other engineering contexts, concurrent engineering or simultaneous engineering, respectively, has yet been very successful in improving engineering output quality and accelerating the overall process lead times in the manufacturing industry. Within the scope of product development processes supported by CAD and Computer-Aided Manufacturing (CAM) tools, concurrent engineering techniques involve multiple designers working on the same end product subdivided into components in parallel. The individual component design models are synchronized at a single point of integration, coordinated by a case manager or case team, respectively, depending on the project's size. Case workers are responsible for the overall design process and frequently check the aggregated contributions for inconsistencies and abidance by the overall product design, also emitting feedback to the individual design teams in order to prevent deviant components prematurely. Since the artifacts produced by individual and possibly remote design teams in concurrent development processes are integrated and combined at some point in time, concurrent development processes have to be seen as a special form of a collaboration scenario.

Transferring such a scenario to the software engineering domain requires the software designers to agree on a common set of design standards and data interchange formats [7]. Since we envision business software development to be shifted to the next consequent level of abstraction, MDD and the related OMG standards briefly introduced in the following section potentially provide the properties required for a concurrent software development process.

4. METAMODELING, MOF, AND XMI

In order to realize a collaborative, concurrent, and model-driven software development process for business applications and business components, we need a set of common standards for software model representation. The advent of the OMG standards *UML 2.0*, the *Meta-Object Facility* (MOF), and *XML Metadata Interchange* (XMI) now paved the way for standardized computer-aided software design using semantically rich and formally specified UML models. The OMG subsumes their current standardization efforts in that context under the term *Model Driven Architecture* (MDA). However, using the MDA especially for developing business applications still bears many problems as discussed in [25]. The wide range of the business domain necessitates systematic adaptations and extensions of the applied modeling standard (i.e. UML in the MDA context).

Metamodeling

The underlying motivation for metamodeling within the context of MDA is analyzed in [3], and the authors predict both a short-term and a long-term boost in software process productivity ignited by using software models as primary artifacts being developed by humans. Using models immediately raises the bar of abstraction in software development itself, and in the long run software assets expressed in models provide extended reusability by capturing conceptual design and platform-independent solutions rather than technical implementations of business problems [9].

Metamodeling in general constitutes the discipline of defining models of models for certain purposes or domains. MOF as a metamodeling language (level M3 in Fig. 3, cf. [20]) provides a set of common model elements to build more specific metamodels upon, such as the UML metamodel (M2 level) and related metamodels [23]. Usually, for most domain-specific modeling problems it would be inefficient to define an all new MOF-based M2-language, but rather reasonable to extend the UML metamodel for business purposes [24].

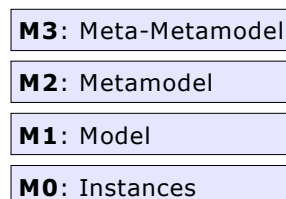


Fig. 3: MOF Layer Model

UML Profiles

Together with metamodeling techniques, extension mechanisms on the UML metamodel layer (M2) form the basis of domain-specific adaptations of the modeling language. UML Profiles, for example, make use of the following extensibility concepts:

- Stereotypes,
- Tagged Values, and
- Constraints.

In essence, stereotypes are metaclasses and tagged values in UML 2.0 represent attributes of metaclasses. Constraints can be added, but existing UML constraints must not be loosened in any profile [22].

XML Metadata Interchange

In order to guarantee interoperability of UML and MOF models designed by individual design teams, XMI provides a standardized *Document Type Definition* (DTD, in XMI 1.2 [18]) or *XMLSchema* (in XMI 2.0 [21]), respectively, for describing model information in an unambiguous formal manner. The standard thus facilitates the exchange, integration, and comparison of models generated by different tools in different organizations.

MDD and particularly the entailed standards now provide the enabling information technology for our reengineered, concurrent software development process, according to the well-known principles of Hammer and Champy who emphasize the role of IT as an enabler for improving business processes [10].

5. THE C³ SOFTWARE DEVELOPMENT PROCESS

Since the primary focus of this paper is on making use of MDD techniques and standards to boost productivity and quality in software development processes, these aspects will particularly be stressed in the following adumbration of the process. As the previous sections described and the working title of the process model implies, the three C's (Concurrency, Collaboration, and Components) form the pillars of the envisioned process architecture.

In Fig. 4 the overall collaborative development scenario for business components is depicted. The main feature of the overall architecture is the domain repository containing domain-specific metadata and components accumulated by the business engineering and software development projects concerning a particular domain (cf. Fig. 1). XML database technology helps to realize those repositories and integrates XMI-based model specifications easily. The final frontier is then the automatic code generation from those XMI-based software and domain models, raising the level of abstraction for business software development.

For describing and explaining our methodology as a whole, we utilize Jayaratna's *Normative Information Model-based Systems Analysis and Design* (NIMSAD) framework [14]. The framework structures a methodology in four major parts:

1. problem situation (methodology context),
2. intended problem solver (methodology user),
3. problem solution process (actual methodology), and
4. evaluation of elements 1-3.

In design science, i.e. the section of information systems (IS) research dealing with the development of (business) software methodologies as problem-solving processes for organizations, the evaluation aspect is particularly stressed alongside with the fact that there has to be a relevant problem situation in the first place in order to trigger methodology research activities [12].

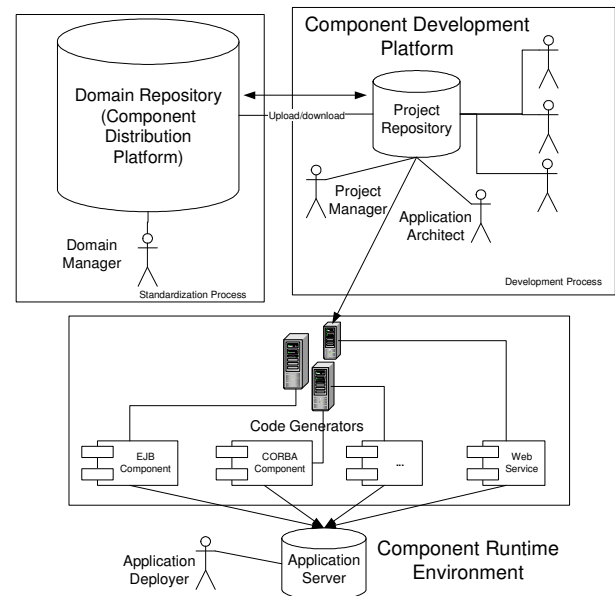


Fig. 4: Collaborative Development Environment

Problem Situation

Building applications for business purposes in a certain business domain, e.g. financial services, constitutes the context for our outlined C³ development process. In particular those business processes that need to be supported or enabled by the business application account for the problem situation and lay the basis for the business requirements and use cases specifying the projected system.

Intended Problem Solvers

The primary methodology users are also depicted in Fig. 4 and play major parts in our role model for the C³ process:

- **Domain Manager:** The domain manager is the designated gatekeeper of business domain information, i.e. components, patterns, and domain ontologies, which are stored, archived, and published on the domain repository platform.
- **Project Manager:** The project or case manager is the integral part in our intended concurrent software design process. He and his supporting tools make sure that the collaborative design effort among different teams produces a consistent end product and that feedback messages are issued in time to avoid subsequent misfit of individual components.
- **Application Architect:** These software architects in the business application area have to transform the

artifacts of requirements engineering and system analysis into a basic structure for integrating functional software building blocks, i.e. components.

- **Component Developer:** The job of the individual developer is to further elaborate on the semantic specification of a business component in the context of a given architecture and dependencies on other components.
- **Application Deployer:** After the actual software components have been generated for a specific component platform, they need to be deployed—and possibly be adapted—in the context of the overall business application. The application deployer’s job is to get the application adapted to the intended business environment and eventually get it up-and-running.

Problem Solution Process

The fundamental steps of a software development process are expressed in terms of phases. As we want to focus especially on the actual concurrent component-based software development and model-driven design processes, in this paper preliminary phases like methodology engineering, domain engineering, business engineering, and requirements engineering are not further elaborated on.

- **Standardization Phase:** The standardization process revolves around the domain metadata repository bearing component models and domain-specific metamodels and ontologies. The repository provides an interface to search for the domain software assets archived, which can then be downloaded onto the project repository that only contains those model elements vital for the business software system currently developed in this particular project. Vice versa, newly developed component models and domain-specific model elements can be uploaded and categorized for further reuse in future projects.
- **Software Development Phase:** The core ideas of our concurrent development process for business software are reflected in the model-based design process. Prior to distributing individual component modeling tasks to different teams, the overall application architecture needs to be defined, among other things as a basis for comprising consistency checks and component integration tests.
- **Model Design:** As indicated in Fig. 4, the component developers and development teams collaboratively work on one business application architecture represented in the XMI standard in one design database—the project repository. This single point of integration enables frequently recurring consistency checks against the outlined architecture and among the individual components. In fact, feedback messages are issued by the supervising project management team responsible for the overall design process (cf. Fig. 4).
- **Code Generation:** Ideally, all facets of the specified business application can be modeled with UML or

XMI means, respectively, on a platform independent level. If that is the case, platform-specific code generation tools are then able to transform the models into deployable software components following the Enterprise Java Beans or CORBA specifications, for instance. In practice, the goal of fully automated code generation is very hard to accomplish, and generating as much executable code as possible is a more realistic aim, at the moment.

- **Application Deployment:** After the generation stage and possible human code completion, the software components should be ready to be deployed on a given application server according to the modeled architectural framework.

Evaluation and Process Metrics

A crucial aspect often neglected by software development methodologies is to measure the performance of particular instances of the process. For our approach a certain set of metrics seems appropriate to achieve this goal, comprising for example:

- degree of component reuse
- share in automatically generated code
- overall project lead time from inception to deployment

Those and other metrics have to be implemented in the collaboration support tools in order to realize online process control and benchmarking as well as post-project evaluation.

Tool Support

The efficient realization of our process heavily relies on appropriate tool support. XMI represents the basis for both design tool interaction and information interchange. The complete XMI-based model then also serves as the basis for automatic code generation and documentation. By implementing online performance measurement components, the overall project progress can be analyzed and managed.

Preliminary Proof of Concept

As a proof of concept for our concurrent component modeling process we applied a very simple open source-based architecture consisting of a conventional CVS (Concurrent Versions System) repository integrating the UML models in XMI format. Since XMI files, like standard XML files, are text-based the basic *diff*-function of CVS is applicable and supports the project manager in his reviews of the models.

6. RELATED WORK

As stated in [14] there are thousands of methodologies or process models out there on the market already. Generic object-oriented frameworks, like the *V-Model*, *OPEN*, and the *Unified Process* offer building blocks or process components for instantiating tailored process models for different organizations and software projects. All three of the approaches mentioned above strive for composition-

based reuse as a means of enhancing development process productivity and quality. MDD or code generation techniques are not yet included in the set of default process building blocks.

There are also more specific component- or reuse-oriented methodologies, like *SELECT Perspective* [1], *Catalysis* [8], *UML Components* [6], and *KobrA* [2], where the latter is the only process model explicitly advocating MDD.

According to the state-of-the-art revealed by our investigations there is no established methodology combining the discussed advantages of component-based, collaborative, and concurrent software engineering in a model-driven process.

7. CONCLUSIONS

Aiming at increased productivity, quality and flexibility in business application development, our methodology introduces the combination of three main features—the three Cs: component-orientation, collaboration, and concurrent engineering. By implementing these principles we envision improved software process performance in terms of time-to-market, reuse, and automated code generation aspects.

In order to verify the superiority and acceptance in real world development scenarios, we are planning to discuss and later implement larger parts of the C³ process in several IT service companies in southern Germany. Moreover, a domain repository for financial service providers is intended to be established in cooperation with two large banks—also in southern Germany. Building code generator frameworks for different component platforms is projected as well. All those efforts are embedded in a large state-funded project called **CollaBaWü**, spanning three years of intensive research/industry collaboration and knowledge transfer.

8. REFERENCES

- [1] Allen, P., and Frost, S., *Component-Based Development for Enterprise Systems*, SIGS Books, 1998.
- [2] Atkinson, C. et al., *Component-Based Product Line Engineering with UML*, Addison-Wesley, 2002.
- [3] C. Atkinson, and T. Kuehne, “The Role of Metamodeling in MDA”, In: *Bezevin, J., and France, R. (eds.): Workshop in Software Model Engineering*, 2002.
- [4] L. Augustin, D. Bressler, and G. Smith, “Accelerating Software Development Through Collaboration”. In: *Proceedings of the 24th International Conference on Software Engineering (ICSE-02)*, 2002, pp. 559–566
- [5] T. Biggerstaff, and C. Richter, “Reusability Framework, Assessment, and Directions”, In: *Software Reusability: Vol. 1, Models and Concepts*, ACM Press, 1989, pp. 1-17.
- [6] Cheesman, J., and Daniels, J., *UML Components*, Addison-Wesley, 2000.
- [7] P. Dewan and J. Riedl, “Toward Computer-Supported Concurrent Software Engineering”. In: *IEEE Computer* 26 (1993), No. 1, pp. 17–27.
- [8] D’Souza, D.F., and Wills, A.C., *Objects, Components, and Frameworks with UML: the Catalysis Approach*, Addison-Wesley, 2001.
- [9] Flater, D., “Impact of Model-Driven Standards”, In: *Proceedings of the 35th Hawaii International Conference on System Sciences*, 2002.
- [10] Hammer, M., and Champy, J., *Reengineering the Corporation – A Manifesto for Business Revolution*, HarperCollins, 2001.
- [11] Herzum, P., and Sims, O., *Business Component Factory – A Comprehensive Overview of Component-Based Development for the Enterprise*, OMG Press, Wiley Computer Publishing, 2000.
- [12] Hevner, A.R. et al., “Design Science in Information Systems Research”, *MIS Quarterly*, Vol. 28, No. 1, pp. 75-105, March 2002.
- [13] Jacobson, I., Booch, G., and Rumbaugh, J., *The Unified Software Development Process*, Addison-Wesley, 1999.
- [14] Jayaratna, N., *Understanding and Evaluating Methodologies: NIMSAD, a Systematic Framework*, McGraw-Hill, 1994.
- [15] A. Korthaus, and T. Hildenbrand, “Creating a Java- and CORBA-Based Enterprise Knowledge Grid Using Topic Maps”, In: *Cheung, W.K., and Ye, Y. (eds.): Proceedings of the Workshop on Knowledge Grid and Grid Intelligence*, Halifax, Canada, Oct. 13 2003, pp. 207-218.
- [16] Korthaus, A., *Komponentenbasierte Entwicklung computergestützter betrieblicher Informationssysteme*, Informatikstechnologie und Ökonomie, Peter Lang, 2001.
- [17] Object Management Group, *Common Facilities RFP-4, Common Business Objects and Business Object Facility*, Request for Proposal, OMG Document Number cf/96-01-04, 1996.
- [18] Object Management Group, *XML Metadata Interchange (XMI) 1.2*, Specification, OMG Document Number formal/02-01-01, 2002.
- [19] Object Management Group, *UML Profile for Enterprise Distributed Object Computing*, Specification, OMG Document Number pct/02-02-05, 2002.
- [20] Object Management Group, *Meta Object Facility (MOF) 1.4*, Specification, OMG Document Number formal/02-04-03, 2002.
- [21] Object Management Group, *XML Metadata Interchange (XMI) 2.0*, Specification, OMG Document Number formal/03-05-02, 2002.
- [22] Object Management Group, *UML 2.0 Superstructure Final Adopted Specification*, Working Document, OMG Document Number ptc/03-08-02, 2003.
- [23] Object Management Group, *UML 2.0 Infrastructure Final Adopted Specification*, Working Document, OMG Document Number ptc/03-09-015, 2003.
- [24] M. Voelter, “Metamodellierung”, Working Paper, Heidenheim, Germany, 2004.
- [25] A. Wegmann, and O. Preiss, “MDA in Enterprise Architecture? The Living System Theory to the Rescue...”, In: *Proceedings of the 7th IEEE International Enterprise Distributed Object Computing Conference (EDOC’03)*, 2003