Towards Specialization of the Contract-Aware Software Development Process

Anna Derezińska and Przemysław Ołtarzewski

Institute of Computer Science, Warsaw University of Technology,
Nowowiejska 15/19, 00-665 Warsaw, Poland
A.Derezinska @ii.pw.edu.pl

Abstract. The contract-based software implementation improves accuracy and verification capabilities of business information systems. This paper promotes contract identification in early phases of the software development and defining contracts within models. Design and implementation artifacts that are responsible for system functionality and system constraints are transformed during the consecutive development phases. Combination of the Model Driven Engineering paradigm and Design by Contract ideas constitutes the Contract Aware Software Development process (CASD). The approach is specialized for system modeling in UML, contracts specified at model level in Object Constraint Language (OCL), and the final implementation in the C# language. The specialized process is supported by the tool transforming the models with associated contracts into the corresponding programs with contracts at the code level.

Keywords: software development process, Model Driven Engineering, UML, contracts, OCL.

1 Introduction

An apparent discrepancy between a system specification and its implementation is a common problem arisen in business systems. A tendency to overlook system constraints can be especially observed during requirement analysis and software modeling and design. The constraints are often modified, as in the evolution of enterprise systems. These error-prone phenomena result in increase of the number of system faults and high costs of a software correction and maintenance.

The answer to these problems lays in using contracts through the whole software development process, including especially modeling activities. Contracts can be understood in general as conditions that must be true for the whole life time of a system unit, or be satisfied at certain time points of a unit behavior. A unit is represented by different kinds of models or software descriptions.

Design-by-contract (DbC™) [1] is a technique that relies on the specification of contracts and is primarily used at the code level. However, the code contracts originate from constraints specified in business rules and software requirements. Consequently the contracts could already be expressed in models at the system design level.

M. Bajec and J. Eder (Eds.): CAiSE 2012 Workshops, LNBIP 112, pp. 494–501, 2012.

This paper is focused on the application of the contract-based methodology combined with model-driven engineering [2]. We propose a general approach, so-called Contract-Aware Software Development (CASD) process [3], in which business system constrains are specified as contracts and applied to system models. Next, the models with associated contracts can be refined and transformed into the code level.

This generic approach can be specialized towards different modeling languages, e.g. UML, and various contract technologies. A language used for formulating of constraints in UML models is the Object Constraint Language (OCL) [4,5]. However, in order to achieve the anticipated objectives it is indispensable to provide an infrastructure supporting the approach. Contract to code transformation is integral to the way the methodology is applied. The contract implementation effort is moved to the discussed transformation tool. Therefore, the code is consistent with the specification. Moreover, the labor required for the contract maintenance can be lowered.

In the next Section, the main issues of the Contract-Aware Software Development process are specified. Section 3 presents a solution adapting the process to selected technologies. In further sections we discuss how some ideas of the CASD process are applied with the tool support and related to other work. Section 6 concludes the paper.

2 Contract-Aware Software Development

Contract-Aware Software Development (CASD) is an approach to a generic process that combines features of the contract-based and model-driven development. The preliminary draft of the process was presented in [3,6]. The main phases of the process and its basic artifacts are illustrated in Fig. 1.

Dual artifacts encounter in various phases of the process, from the analysis to the implementation one. The artifacts shown on the left hand side correspond to a system functionality, whereas the opposite elements represent constraints and contracts.

The artifacts within a development phase are related by a *constrain* dependency. It represents the coupling between the system functionality and the corresponding contract at the same abstraction level. The inter-phase relations are defined by *transform* and *trace* dependencies. *Transform* dependencies illustrate the gradual refinement of the artifacts during the development process. The latter dependencies are responsible for defining traceability to the original artifacts. During the system development and maintenance, changes in a contract made at the i-th level of abstraction should also be incorporated to the previous level. This fact denotes the need for the round trip engineering, which refers to the contracts as well as to the functional artifacts. Both types of the artifacts should be transferred into the next abstract level in a common step.

In general, the basic concepts of the process are defined as a finite sequence $< L_1, \delta_1, L_2, ... \delta_n, L_{n+1} >$ including two kinds of elements. The first kind of elements $L_i = (F_i, C_i, \lambda_i)$ is a tuple denoting an ith level of a system abstraction, where:

i = 1…n+1 is a number of a considered level of system abstraction,

ith level is more abstract than (i+1)th level,

F_i is a finite set of functionality artifacts at the ith level of system abstraction,

C_i is a finite set of contracts at the i[th] level of system abstraction,
$\lambda_i : C_i \rightarrow F_i$ is a function representing the *constrain* dependencies.

The second kind of elements in the process is a transformation function between two adjacent levels of system abstraction: $\delta_i : 2^{F_i} \times 2^{C_i} \rightarrow 2^{F_{i+1}} \times 2^{C_{i+1}}$, where 2^X denotes a power set of set X and i =1…n.

The transformation function satisfies the following implication:

$$\forall c_i \{((\lambda_i(c_i) = f_i) \wedge \exists \delta_i((\delta_i(A_i, B_i) = (A_{i+1}, B_{i+1})) \wedge (f_i \in A_i) \wedge (c_i \in B_i))) \quad (1)$$
$$\Rightarrow \exists \lambda_{i+1}((\lambda_{i+1}(c_{i+1}) = f_{i+1}) \wedge (f_{i+1} \in A_{i+1}) \wedge (c_{i+1} \in B_{i+1}))\}$$

where:

$f_i \in F_i, f_{i+1} \in F_{i+1}$ are functionality artifacts from the i[th] and i+1[th] levels,

$c_i \in C_i, c_{i+1} \in C_{i+1}$ are contracts from the i[th] and i+1[th] levels,

$A_i \subseteq 2^{F_i}, A_{i+1} \subseteq 2^{F_{i+1}}, B_i \subseteq 2^{C_i}, B_{i+1} \subseteq 2^{C_{i+1}}$ are subsets of functionality artifacts and contracts from the i[th] and i+1[th] levels, respectively.

The formula (1) depicts preservation of *constrain* dependencies between the corresponding artifacts in a transformation. If a pair of artifacts is related at i[th] level, then after transformation, their resulting artifacts at (i+1)[th] level should also be related.

The identified phases do not suggest that the process follows the rules of the waterfall process. The succession of phases shown in Fig. 1 focuses on the main idea recognizing the fundamental artifacts of the process. Possible back dependencies and the overlapping of phases are omitted in the figure.

The dependencies within the process, as well as the automated transformation of functionality artifacts and contracts support system evolution, especially for changing business requirements. This corresponds to the intrinsic postulate of the agile approaches and enterprise system development. In CASD, changes in the functionality and/or in constraints can be propagated through the appropriate process levels.

3 CASD Specialization for UML, OCL and C#

The general idea of the generic CASD process can be specialized by applying various methods and techniques for selected artifacts and their transformations.

The definition and maintenance of the *constrain* dependencies (Fig.1) is important for the process realization. It is recommended to have a tight coupling between artifacts of the both sides of the process, preferably tool supported. Otherwise the additional overhead and manual effort would discourage the concurrent development of both types of the artifacts. Moreover, certain contracts might be easily missed due to an erroneous omission by a developer. Another recommendation is automating of the intra-phase transitions and maintenance of the trace dependencies.

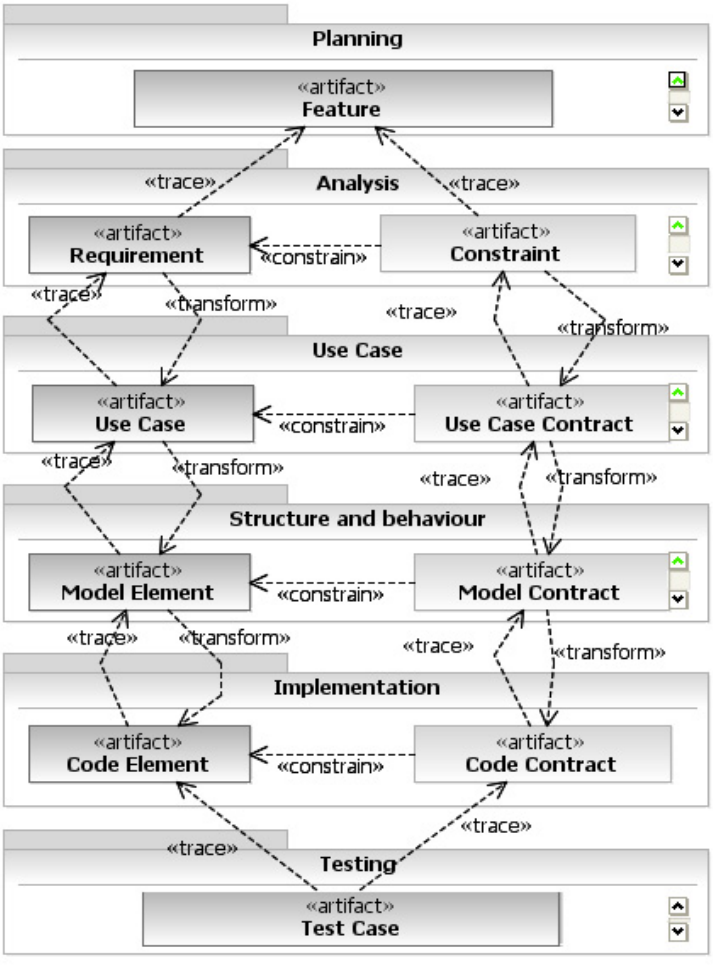

Fig. 1. The Contract Aware Software Development process

System models can be designed in UML, especially class diagrams accompanied by logical constraints specifying business contracts at the model levels [7]. They can be specified in OCL [4,5]. OCL is a declarative specification language. OCL expressions can be used for defining class invariants, pre- and post-conditions of operations, and other constraints associated with model artifacts.

UML is a language that can be applied at different levels of abstraction and used for different purposes. Model refinement towards a selected Domain Language and desired technology can be accomplished by application of model profiles.

Apart from model to model transformations aimed at the gradual model refinement, models can be translated into the source code, or at least to a correct but incomplete subset of the final code. One of many possible solutions, which can be used for the implementation of business systems, is the C# programming language.

A crucial part of such a specialized CASD process is refinement of UML models and transformation of OCL into contracts at the code level. The C# language does not directly include concepts of contracts, but we can utilize the Microsoft Code Contract library [9] that implements contracts. The library integrates with the .NET 4.0 platform and supports C# and Visual Basic programming languages.

The basic part of the library is the *Contract* class including a set of static methods that enable description of contracts. Invariants can be specified for classes, structs and interfaces. Pre- and post-conditions can be associated with constructors, methods, overloaded operators, type conversions and accessor methods of properties, events and indexers. A special static event defined by the *Contract* class is called in case of contract invalidation. Transformation of OCL invariants of classes and pre-, post-conditions specifying operations can be based on the *delegate* concept of C#. The details of this transformation are omitted due to brevity reasons.

4 Transform OCL Fragments into C# - T.O.F.I.C Tool

The core part of the CASD process constitutes the transformation of models and their contracts into the executable applications including the corresponding contracts. The transformation should satisfy preservation of dependencies defined in Sec. 2. To put this into practice the transformation should be automated and the modeling activities assisted in a friendly manner.

The specialized process can be supported by the T.O.F.I.C. tool (*Transform OCL Fragments into C#*) [6]. This tool extends the CASE tool - the IBM Rational Software Architect [10] with the C# modeling and code generation capabilities. The tool consists of a set of plug-ins of the Eclipse framework that creates so-called Eclipse *feature*. The main characteristics of the preliminary prototype version of the tool were described in [11].

The current version of T.O.F.I.C. 1.1.7 was considerably extended in comparison to its prototype. It covers new functionality (e.g. most of OCL, many structures of C#, contracts in C#) and improves its ergonomic features.

The UML profiles are used for modeling of Domain-Specific Languages (DSL), in this case C# concepts and code mapping units. The approach could be adjusted to the modeling of various business information systems according to their needs.

The current version of T.O.F.I.C. was enhanced with the profile tooling. It includes GUI elements that support visualization and editing of the stereotyped elements of a model. Rapid prototyping and assigning of stereotypes to the selected model elements is realized by the C# Action Tool (CAT). It extends the palette of modeling menus with the appropriate view. Various buttons with graphical markers can be used to assign a selected stereotype and visualizes this distinction in the diagram.

The tool also facilitates the C# code generation from a refined UML model and OCL constraints. The generated code is extended with the C# implementation of the standard OCL library. Transformation of OCL contracts is realized using the Visitor design pattern. The code generation is initiated with the creation of the Abstract Syntax Tree (AST) of an OCL expression based on its text representation in a model.

Next, according to the appropriate transformation rule, the nodes of the tree are visited in a given order. During the tree traversal the corresponding C# code is generated and stored in the related compilation unit.

Another new feature, the most significant in the context of the CASD process, is usage of contracts in the target code of OCL. Expressions of OCL are translated into the corresponding method calls of the Microsoft Code Contracts library [9].

Application of the library benefits from an existing contract solution that integrates with the Visual Studio - the commonly used development framework. However, it has also negative consequences due to limitations of the library.

The .NET 4.0 platform supports defining of contracts and application of the dedicated namespace. Verification of contracts is realized by the additional library that should be installed within the Microsoft Visual Studio environment. The library is supported by the tools available via a command line as well as using a GUI extension. Contracts are verified during a project building and in the runtime.

A limitation of the library concerns reaction to a contract invalidation. A default reaction on a raised exception can be modified by a developer by implementing the own class for handling contracts in the runtime. Furthermore, there is no information which instance is responsible for a contract invalidation. Therefore the exception triggers handling of all delegates associated with the exception.

The usability of the approach, the maturity of the extended tool and the impact of adopting contract code generation in the software development process were evaluated in an experiment. The experiment was conducted by students of an advanced course of software engineering. The participants of the experiment cooperated on the development of a common system, eliciting and specifying requirements of particular system modules. The system simulated business and control processes of an airport. The requirement specifications took into account various system constraints. Next, the constraints were included in the description of use cases.

The team members swap the requirement specifications among others. UML models were designed according to the obtained specifications. The general models were supplemented with appropriate contracts written in OCL and refined to the C# code models with assistance of CAT. Then, the C# projects were generated from the refined models by the T.O.F.I.C. tool. The OCL contracts were transformed to the corresponding code using the Microsoft Code Contracts library.

In the experiment the impact of the automatic code generation from models with contracts to the software development was examined. The results of the experiment were evaluated in two ways. The models, other intermediate artifacts, and final applications with tests were examined by hand and using static and dynamic verification tools. In addition, the participants filled in a questionnaire after the experiment.

The general evaluation of all results confirmed the improved consistency between the code and this level of specification that was expressed in refined models and OCL constraints. In the questionnaire, the most of participants admitted that they would be utilizing the T.O.F.I.C. tool to a project development in the future, assuming the selected improvements were incorporated.

The key obstacles of the approach are problems of consistency between preliminary business rules described in requirements and use cases on the one hand and

contracts at the model level on the other hand, the increased effort required during refinement of models, and the obligation of knowledge of OCL.

Another alternative to the process specialization could be combining UML models with constraints written in a target implementation language, e.g. Java or C#. This kind of specification of business rules requires validation of constraints at the model level, in order to be of any practical use.

5 Related Work

Design by Contract principles [1] are first of all applied at the code level, as in the Eiffel language [12]. However, there is a lack of tools dealing with contracts on both abstract levels, as models and code.

There are many tools that support OCL [13,14], but the most of them do not generate code from OCL constraints. In [15] such tools were compared taking into account their contract generation capabilities. All these tools, apart from T.O.F.I.C., generate Java code from OCL. There was announced a vision of a potential, preferred OCL tool, but it is still a future work [16].

Using Java as a target language, we can utilize Dresden OCL [17] in a specialized process with OCL. This tool transforms OCL expressions into aspects of AspectJ. OCL constraints used in MDE towards Corba and Java can also be found in [18,19].

Support for the C# code generation has been incorporated into several CASE tools, but none of them supports contract-based approach for C# at the generated code level.

6 Conclusions

The paper presents an approach that combines model-driven development principles with advantages of contract utilization. The approach was applied using selected modeling, specification and implementation languages. The early definition of contracts, as recommended in the CASD process, focuses the attention of developers on the constrains and their verification. They should reflect business rules of a system. OCL constraints are transformed to code contracts supported by a library in order to move the benefits of design by contract approach to the modeling level.

The preliminary experiment concludes that the tool supported contract evaluation combined with the model-driven methodology could improve the application accuracy and testability. The applicability of the methodology depends strongly on the convenient tool support, which has to be further enhanced. The critical issues also remain completeness of system constraints implemented as contracts.

As far as the obstacles in the contract utilization with T.O.F.I.C. are concerned, the contract library used in the tool could be substituted by another contract solution, or there will be available an improved version of the Microsoft Code Contracts library.

The solution can be extended with code generation from the dynamic models, e.g. state machines. State invariants and guard conditions can also be transformed into the appropriate code contracts.

References

1. Meyer, B.: Object-Oriented Software Construction, 2nd edn. Prentice-Hall (1997)
2. France, R., Rumpe, B.: Model-driven Development of Complex Software: A Research Roadmap. In: Proc. of Future of Software Engineering at ICSE 2007, pp. 37–54. IEEE Soc. (2007)
3. Derezińska, A., Ołtarzewski, P.: Business Software Development Process Combining Model-Driven and Contract-Based Approaches. In: Jałowiecki, P., Łuasiewicz, P., Orłowski, A. (eds.) Information Systems in Management XI, pp. 7–17. WULS Press, Warsaw (2011)
4. Object Constraint Language (OCL) (March 15, 2012), http://www.omg.org/spec/OCL/
5. Warmer, J., Kleppe, A.: The Object Constraint Language: Getting Your Models Ready for MDA. Addison-Wesley Professional (2008)
6. Ołtrzewski, P.: Software Development Using Contracts by Example of the T.O.F.I.C Tool. Master Thesis, Inst. of Computer Science, Warsaw Univ. of Technology (2011) (in Polish)
7. Neumarite, L., Ceponiene, L., Vadrickas, G.: Representation of Business Rules in UML&OCL Models for Developing Information Systems. In: Stirna, J., Persson, A. (eds.) POEM 2008. LNBIP, vol. 15, pp. 182–196. Springer (2008)
8. Frankel, S.: Model Driven Architecture: Appling MDA to Enterprise Computing. Wiley Press, Hoboken (2003)
9. Microsoft Code Contracts (March 15, 2012), http://research.microsoft.com/en-us/projects/contracts
10. IBM Rational Software Architect (March 15, 2012), http://www-01.ibm.com/software/awdtools/swarchitect/
11. Derezińska, A., Ołtarzewski, P.: Model-Driven Engineering Support for Building C# Applications. In: Sobh, T., Elleithy, K. (eds.) Innovations in Computing Sciences and Software Engineering, pp. 449–454. Springer (2010)
12. Eiffel Software: An Eiffel Tutorial (March 15, 2012), http://docs.eiffel.com/
13. Chimiak-Opoka, J., Demuth, B., et al.: OCL Tools Report Based on the IDE4OCL Feature Model. In: Proc. of International Workshop on OCL and Textual Modeling, col. Tools Europe (2011)
14. Toval, A., Requena, V., Fernandez, J.L.: Emerging OCL tools. Journal of Software and System Modeling 2(4), 248–261 (2003)
15. Derezińska, A., Ołtarzewski, P.: Code Generation of Contracts Using OCL Tools. In: Borzemski, L., et al. (eds.) Information Systems Architecture and Technology, Web Information Systems Engineering, Knowledge Discovery and Hybrid Computing, pp. 235–244. Publishing House of Wroclaw University of Technology, Poland (2011)
16. Chimiak-Opoka, J., Demuth, B.: A Feature Model for an IDE4OCL. In: Proc. of International Workshop on OCL and Textual Modeling (2010)
17. Dresden OCL, (March 15, 2012), http://reuseware.org/index.php/DresdenOCL
18. Coronato, A., De Pietro, G.: Formal design and implementation of constraints in software components. Advances in Engineering Software 41, 737–747 (2010)
19. Dan, L., Danning, L.: Applying Model Driven to Software Development: a University Library Case Study. In: Proc. of the 3rd International Conference on Communication Software and Networks, ICCSN, pp. 179–183. IEEE Comp. Soc. (2011)