

USING CONTRACTS FOR DEVELOPING AND TESTING SOFTWARE SYSTEMS

György ORBÁN

Faculty of Informatics, Eötvös Loránd University
Budapest, Hungary
orbangyorgy@caesar.elte.hu

and

László KOZMA

Faculty of Informatics, Eötvös Loránd University
Budapest, Hungary
kozma@ludens.elte.hu

ABSTRACT

Software development techniques need to be equipped with quality views too. Quality factors like reusability, extendibility, compatibility are important factors in component based development. To support component based development and to build more reliable software systems new techniques, methods and tools are needed.

In our paper we discuss recent research areas, available tools which support the “design by contract” methodology. The reliability of the components can be increased using the contracts introduced by Bertrand Meyer. The number of tools supporting contract based development is increasing. From the available tools which support contract based development we choose two one in Java (Contracts for Java) and one in .Net (Code Contracts) environment. We also compare these tools and for this we choose some main point of views like do they support static code analysis, dynamic checking or can they help the testing processes. The educational aspects of contracts are discussed as well.

Keywords: Components; Component Based Design; Software Testing; Contracts; Design by Contract

1. COMPONENT BASED DESIGN

If we talk about component based development we can talk about the development of each component and about the development of the system from components. We will use the definition of H-G Gross about the component:

“A component is a reusable unit of composition with explicitly specified provided and required interfaces and quality attributes that denotes a single abstraction and can be composed without modification.” [13]

As we can see based on Gross’s component definition one of the important attributes of a component is its reusability. Using components can make the development process faster. If there are prebuilt components, the system can be built from them, maybe some glue code, or a middleware is necessary, so the components can communicate with each other. For the communication

between the components and between the components and the environment well defined interfaces are necessary.

In the paper we will see, how the first level contracts can help to define the interfaces, but the main parts in the paper are related to the behavior level (second level) contracts.

The interfaces which belong to a component are categorized into two types. The first type is the *necessary* interface, which is needed by the component to work properly. The second type is the *provided* interface where the component provides its services.

The KobrA approach addresses the problem that the Component Based Software Engineering (CBSE) is typically focused on the component implementation. The focus in the KobrA method is on the entire software development process, not just the implementation and deployment, by adopting product-line strategy, the creation, maintenance and deployment of components [1]. It is very important to use modeling languages like UML. Creating the UML model of the component can make the analysis and design processes component oriented. After the model based design the components can be implemented using any of the available component models CORBA [19], DCOM [25], EJB [26, 27] etc.

The KobrA approach can be used for educational purposes to because it tries to merge the benefits from more software development methods OMT [23], Fusion [5], ROOM [24].

There are three main dimensions in the KobrA development process. From these three dimensions there are six basic orientations for the development. These are composition/ decomposition, abstraction/concretization, generality/specialization.

When designing a system it will be decomposed into smaller parts, and when each of the parts are implemented the whole system can be composed from the components. Using abstraction and concretization can help to understand and develop the software system starting from an abstract model and converging to a concrete

implementation. First a component can be developed in a generic way and what it is needed a more target specific version can be created from it.

Component models

Software systems can be built from different functional and logical components. Using component models the components can be implemented separately in different developer teams. To build the system software support is needed. This support can be a middleware which runs on top of the Operating System and provides some functionality for the components (maybe written in different programming languages), so they can be connected together and they can communicate with each other.

There are many well known component models like Common Request Broker Architecture (CORBA [19]), Distributed Component Object Model (DCOM [25]), Enterprise JavaBeans (EJB [26, 27]), but there are newer architectures too like ICE from Zeroc [15].

2. DESIGN BY CONTRACT

Design by contract is an approach to design more reliable software. Reliability is defined here as the “combination of correctness and robustness” [17] as Bertrand Meyer suggests it. Reliability is even more important in object oriented programming and in component based software development where there are many reusable parts in the system.

In the development processes classes are defined as “implementations of abstract data types” [18], but these definitions are only attributes and routines. With the usage of contracts the semantic properties can be added to the definitions. With these axioms the properties written down in the specification can appear in the software systems. These can help the developer in the design, development, debugging and testing processes.

Contracts

The theoretical background of the contracts is the correctness proof methods of programs, software. A software system is correct with respect to its specification, so the software correctness is a relative notion. A specification can be expressed by assertions [12, 14, 18].

Let P and Q assertions about program variables and S a statement or a program, then the notation $\{P\}S\{Q\}$ denotes a correctness formula informally meaning that if P is true before the execution of S, then Q is true after execution of S. You can find a very good introduction in Bertrand Meyer’s book [18] to create and introduce assertions into software texts. Several proving methods for sequential and parallel programs have been developed by this time [12, 14, 16, 20]. At our university there are lectures related to these methods. These lectures give a good basic understanding of the contracts and the usage of the contracts in practice. Practicing these methods the

students will be well-skilled for creating contracts during developing reliable software from verified components.

One of the main aim of introducing contracts to decrease complexity [18] because the more complex is a system the more errors can be in the design or implementation. Contracts can help because all the necessary checks related to correctness can be put in the contracts so there is no need to check these properties in other places in the implementation.

When defining the behavior of the system with contracts there are three main variants of them which are preconditions, post conditions and class invariants. These variants define obligations and benefits related to the participants in the system. If there is a provider and a client in our system the benefit for the provider is an obligation for the client and vice versa. At the provider side in the preconditions there are the obligations for the client. If the client wants some services it needs to satisfy the preconditions, but if the preconditions are satisfied the post conditions assure that the provider makes its task correct which is a benefit for the client, but an obligation for the provider. With the third kind of contracts (invariants) we can describe such axioms which must hold through the execution of the routine, or method.

These main three contracts can be used to define the behavior of an object when preconditions, post conditions and invariants are used in the routine implementations. But when bigger and more complex systems need to be built, there is a need for more levels of contracts [3]. These contracts can be used to define not only behaviors in the system, but connection and synchronization related aspects or quality properties too, between software components.

Using contracts in the development process can help to make more correct software but it has documentations values too. The specifications which are identified at the analysis phase of the software development process can be written down by contracts. So it will be easier to check the working system against the specification. The other value it adds to the development and debug processes is that the developer or maintainer can see the specification in the source code when developing or searching for errors or bugs [18].

Four levels of contracts: As in real life contracts can be found in different levels, so the contracts used in software systems can be available in different levels too [3]. The main four levels are syntactic level, behavioral level, synchronization level and quality of service level. On each level there are different techniques for contracts.

On the first syntactic level IDL can be used to define the “connectors” between the components. There are many IDLs that can be used, which one to choose depends on the system environment and the communication methods between the parts.

On the second behavioral level using preconditions, post conditions and invariants the behavior can be defined.

The third synchronization level is needed, because the first two levels do not recognize for example parallel executions. The third level contracts specify the synchronizations between method calls.

The fourth level is for quality of service. This is necessary to write down contracts related to latency or the precision of the result. These attributes can be negotiated between the client and the provider.

Handling the contract violations can be different related to the level. On different levels if a contract is violated the program can behave differently it can ignore the violation, reject it, wait or on the third level negotiation is available too.

In this paper the focus is on the first two levels, the IDLs and mostly contracts related to behavior on the second level. In our tools the contract violation handling is in most cases exception handling.

Contracts in middleware

To connect different components in a system well defined interfaces are needed. In a system where the different components are written in different languages, the necessary interfaces have to be available on all programming languages. For this there are many IDLs (CORBA IDL, Java IDL (Figure 1), Slice [15], etc.). From these IDLs interfaces can be generated for the different components. These interfaces will connect together the software system built from different components.

```
interface MultiplyNumbers{
    void multiply( in int a, in int b, out double result);
};
```

Figure 1: Java IDL

The simple IDL (Figure 1) is an interface description where the multiply method has two input parameters 'a' and 'b' and an output parameter the result.

There are different kinds of middleware's like Babel [2, 8]. Babel was developed to be a common middleware for components implemented in different programming languages. For the supported programming languages an interface could be generated from a SIDL (Scientific Interface Definition Language) so they could communicate with each other through the middleware. SIDL is extended for scientific applications with the support of dynamic multi-dimensional arrays, complex numbers, In-process optimizations, etc. These extensions were necessary for the scientific community to develop applications.

There is also an opportunity to use contracts in the SIDL specifications so the generated interfaces for the different languages will contain the contracts. The usage of contracts in interfaces makes available to check whether

the connected components can interact, communicate with each other.

Support for contracts in different programming languages

There are many programming languages (Eiffel, D, Lisaac, Spec#) which support design by contract in a native way. These languages were designed to support contracts. These contracts are on the second level, so they define the behavior of the component.

Recently there are many third-party tools that can add the opportunity to use contracts in the software implementation phase for programming languages which does not support the contracts in a native way. There are tools for C/C++, C#, Java, Python. In the next chapter there is an introduction of two tools which support contracts in Java and .NET programming languages.

The common part in almost all of these extensions of languages, that they use the keywords *require*, *ensure* and *invariant* for the contracts. The *require* keyword means the precondition, the *ensure* means the post condition and the *invariant* means the class invariants. There are some other special keywords, but these may differ in different programming languages, these are the *old* and the *return* keywords. The *old* keyword refers to the value of a variable when the method execution starts, and the *return* keyword refers to the return value of the method. With these keywords more meaningful post conditions can be written related to the methods behavior.

Contracts and security

If a software system is built from components it is very important to know how these components will work individually or together in a common environment. Contracts can be used for checking the components behaviors. Security properties can be structured in different ways. Like properties relate to security functionality, software quality, monitoring, certification. Based on security properties four different kinds of security related contracts can be identified. These are functional interaction, interaction protocol, security-specific interaction and infrastructure [28]. Using single level contracts to create secure systems is not enough when creating big or complex systems different levels of contracts are necessary.

3. CONTRACTS IN DEVELOPMENT, DEBUGGING AND TESTING

As we see many programming languages support contracts in the development processes. The usage of contracts in software development spreads slowly, but there are many research projects related to develop and create tools which support very well the contract based development.

From the development view contracts can be useful in the source code. There is a connection between Test Driven Development and Contract Based Development. In both

cases the tests and contracts can be written before the actual implementation. The contracts and the tests are created based on the specification. Using contracts the source code contains the specification. Changes in the specification and so in the contracts have immediate impact on the source code.

When contracts document the specification in the source code, it will be available for the later code maintainer, and also if the component with contracts will be reused in another system, makes the search for errors or bugs easier, and helps the component integration into new systems.

In this paper we focus mainly on two (Contracts for Java [4], .NET Code Contracts [6]) tools which support the contract based development. Both tools are under continuous development. There is a short introduction of the tools followed by a comparison.

Java environment

Contracts are supported in Java too. There are many projects (Modern Jass, Contracts4J, jContractor) related to develop contract support for java. These projects were started formerly but there are some projects where the further development stopped for some reasons. There is a project under continuous development the Contracts for Java. Contracts for Java [4] is based on Modern Jass with an enhanced version of compilation model. Their goal is to develop a robust standalone framework dedicated to contract programming in Java. Contracts for Java consists of three parts an annotation processor, an instrumentation agent and an offline byte code rewriter. Contracts can be written into the java source code (Figure 2).

```
interface MultipleNumbers{
    @Ensures({"a>=0", "b>=0"})
    void multiple(int a, int b);
}
```

Figure 2: Contracts for Java

There is a precondition in the example, that the multiplied numbers 'a' and 'b' must be larger or equal to zero.

.NET environment

Microsoft Research develops an extension for the Microsoft Visual Studio which supports contract based development for .NET programs. There is a simple example for the usage of Code Contracts [6].

```
public void AddToList(int num)
{
    Contract.Requires((num > 5) && (num < 10));
    Contract.Ensures(list.Count ==
    (Contract.OldValue(list.Count)+1));

    list.Add(num);
}
```

Figure 3: Simple .Net 4.0 source code with method precondition and postcondition

This simple method (Figure 3) adds a number to a list, but with the precondition and post condition checking the behavior of the method is available. With the precondition (requires) the value of the input number is checked and with the post condition the method behavior is ensured. In this case the method can add the number only once to the list, so in the post condition the count of the list is checked that it was increased by one.

4. COMPARISON OF CONTRACT BASED DESIGN TOOLS FROM DIFFERENT ASPECTS

The tools we compare in this paper is the Microsoft Code Contracts integrated into Microsoft Visual Studio 2010 and the Contracts for Java extension. The main aspects of comparing are how the extensions support static analysis, dynamic checking, the testing processes and documentation, document generation.

Static analysis

Static analysis is performed without executing the software. The analysis can be made by an automated tool, or manually by another developer. Right now we focus on automated tools. Static analysis may include finding coding errors or formal methods to prove properties based on the specification.

Contracts for Java does not support static analysis only dynamic contract checking is available.

Using the Microsoft Visual Studio and the Code Contracts extension it is available to make static analysis during the development process. With static analysis the source code is analyzed at compile time, so finding errors is available before the software execution. Fixing the errors can be made earlier and faster especially if the run environment is big and complex. The static checker can decide if there are any contract violations without running the program. It can check implicit and explicit contracts. Implicit checking contains null dereferences or array bounds checking.

Dynamic checking

Dynamic checking is a runtime activity. The analysis is performed when the program is executing.

Contracts for Java supports only dynamic checking. When a contract is broken a java exception is thrown. There are two techniques for contract checking.

Using the first technique a separate file is generated at compile time for the contracts. The checking of contracts is made with the usage of java agents at runtime. This has some advantages, because the contracts can be handled separate from the source code, which makes easier the later reuse of the contracts. Also the application source code can be built separately.

The other technique is an offline byte code weaving. Where the generated separate contract file is weaved into

the programs byte code before execution, and the contracts are checked runtime, there is no need for java agents, and so there are no Contracts for Java dependencies.

With the Microsoft Code Contracts a binary rewriter modifies the software injecting the contracts so they are checked at runtime.

Documentation (document generation)

The usage of contracts in the source code is already some type of documentation. The contracts contain the specification of the software system, so it can help the developers, and the maintainers to understand the semantics of the software system.

Contracts for Java supports the extension of classes and interfaces with contracts, so it is available to create documentation in the source code with contracts.

Microsoft Code Contracts also supports the documentation in the source code but it is available to generate external documentation from the contracts in the source code. With the document generation it is easier to maintain the documentation related to the software. The modifications in the specification, in the contracts can be updated faster in the additional documentations.

Testing

To support software testing is very important. Contracts can help in the development processes, but they can be used in testing too. Contracts can help in white-box testing. With the usage of contracts more meaningful tests can be created.

The exceptions can help to find errors, because if there is an exception related to the precondition, the client who tries to access a given service made an error, because the input is not appropriate. If there is an exception related to the post condition there is some error in the service provider, so the error investigation should be there. The usage of contracts can help to find the errors easier.

In java environment junit can be used to create unit test. If there is a contract violation a java exception is thrown.

Code Contracts can be connected to support the test phase of a software system with another extension for the IDE, the Pex tool [21]. Pex is an automated parameterized unit test generator for .NET programs. With Pex automatic test generation is available with high code coverage. It is also a research project. These two tools can work together so when Pex is analyzing the source code to generate tests it considers the contracts so there can be more meaningful test cases generated.

Like in Figure 3., there is a precondition which says that the input parameters can be integers between five and ten. With this contract information Pex can generate automatically parameterized unit tests related to the input values.

n	result(target)	Summary/Excepti...	Error Message
0		ContractException	Precondition failed: (num > 5) &...
6	new Test{}		

Figure 4: Automatically generated test cases by Pex

Pex generates automatically test cases with input values 0 and 6 (Figure 4). If the input value is 0 a ContractException is thrown, if the input value is 6 there is no exception, the method can be executed properly based on the given precondition. With these values Pex tries to cover every available execution paths.

5. USAGE IN EDUCATION

There are some courses at our university where the students can learn about the correctness proof methods of sequential and parallel programs, or how to build component based software systems and verify them. During their studies they develop skills for applying development methods and tools for building software systems from verified components. In the lectures the students learn about partial and total correctness of structured programs by Hoare methods,

To create verified software systems formal methods can be used. These methods, like those based on temporal logic or Floyd's and Hoare's methods for proving correctness need the necessary mathematical background and the tools which support formal specification and verification at model or implementation level. Extending these more abstract methods to create correct, verified software systems contract based design tools can add a more practical approach. These are needed to turn an abstract representation of the system into a more concrete representation, for example an executable, deployable component.

To figure out which tools support the best these approaches the comparison of the available tools is needed.

Contracts for is available for free, but it has not so many features like Microsoft Code Contracts and it is a newer project. It is quite easy to setup the development environment and it can be integrated into the Eclipse IDE [9]. With the usage of annotation processing in the Eclipse IDE it is much easier to write contracts, because if there is some missing variable in the contract IDE will indicate it.

In contrast to Contracts for Java, Code Contracts is not free but it supports more techniques static analysis, document generation and more meaningful test cases with Pex.

These tools can help the developers to write more formal specifications with contracts, based on the requirements. They can help to create software systems with better quality. The process of the contract based development could be similar to test driven development, when the development of test cases precedes the implementation.

Contract based development could be made as test driven development, where the developer starts the development with the test cases and after the tests comes only the actual implementation of the methods. Using contract based design the contracts could be created first. After validating these contracts they can be used in the developed system.

Contract based development could be part of software quality lectures or lectures related to software testing also.

6. CONCLUSIONS

There are many continuously developed tools (EiffelStudio [11]) and programming languages (Eiffel [10], D [7], Python [22], etc.) which support the contract based development. Further research is necessary to compare more available tools and functionalities they support. These tools can help during software development, testing including debugging and maintaining. The contracts could be used in the debug code to help find errors.

Building the different components with contracts a built in checking is available for the components. This type of checks can be very useful during integration tests when the components are composed into a software system, in such cases when a component is changed in the system, or if the component has to work in a new environment.

Contract based development should be used to create more reliable and better quality software.

7. ACKNOWLEDGEMENTS

The Project is supported by the European Union and co-financed by the European Social Fund (grant agreement no. TAMOP 4.2.1./B-09/1/KMR-2010-0003).

8. REFERENCES

- [1] Atkinson, C., Bayer, J. and Muthig, D. 2000. **Component-Based Product Line Development: The Kobra Approach**. *SOFTWARE PRODUCT LINE CONFERENCE*, 289–309, 2000.
- [2] **Babel Homepage**: 2010. <https://computation.llnl.gov/casc/components/index.html>. Accessed: 2011-06-19.
- [3] Beugnard, A., Jézéquel, J.-M., Plouzeau, N. and Watkins, D. 1999. **Making Components Contract Aware**. *Computer*, 32, 7 (jul. 1999), 38–45.
- [4] cofoja - **Contracts for Java**: <http://code.google.com/p/cofoja/>. Accessed: 2011-08-31.
- [5] Coleman, D. **Object-Oriented Development. The Fusion Method**. Prentice Hall., 1994.
- [6] **Contracts** - Microsoft Research: 2011. <http://research.microsoft.com/en-us/projects/contracts/>. Accessed: 2011-06-18.
- [7] **D Programming Language**: <http://www.d-programming-language.org/index.html>. Accessed: 2011-09-14.
- [8] Dahlgren, T., Epperly, T., Kumfert, G. and Leek, J. **Babel Users' Guide**., 2009.
- [9] **Eclipse**: <http://eclipse.org/>. Accessed: 2011-08-14.
- [10] **Eiffel Software**: <http://www.eiffel.com/products/>. Accessed: 2011-09-14.
- [11] **EiffelStudio**: <http://www.eiffel.com/products/studio/>. Accessed: 2011-09-14.
- [12] Floyd, R.W. **Assigning Meanings to Programs**. *Proceedings of Symposium on Applied Mathematics*, 19, (1967), 19-32., 1967.
- [13] Gross, H.-G. **Component-based Software Testing with UML**. Springer-Verlag., 2005.
- [14] Hoare, C.A.R. 1969. **An axiomatic basis for computer programming**. *Communications of the ACM*, 12, (oct. 1969), 576-580.
- [15] **Ice**, Zeroc: <http://www.zeroc.com/>. Accessed: 2011-08-31.
- [16] Manna, Z. **Mathematical Theory of Computation**. Mcgraw-Hill College., 1974.
- [17] Meyer, B. **Applying „design by contract”**. *Computer*, 25, 10 (oct. 1992), 40-51., 1992.
- [18] Meyer, B. **Object-Oriented Software Construction, Second edition**. Prentice Hall., 1997.
- [19] Object Management Group **History of CORBA, Technical Report**., 1997.
- [20] Owicki, S. and Gries, D. **An axiomatic proof technique for parallel programs I**. *Acta Informatica*, 6, (1976), 319-340., 1976.
- [21] **Pex, Automated White box Testing for .NET** - Microsoft Research: 2011. <http://research.microsoft.com/en-us/projects/pex/>. Accessed: 2011-06-18.
- [22] **Python zope.interface**: <http://pypi.python.org/pypi/zope.interface/>. Accessed: 2011-09-14.
- [23] Rumbaugh, J. **Object-Oriented Modeling and Design**. Prentice Hall., 1991.
- [24] Selic, B. **Real-Time Object-Oriented Modeling**. Wiley., 1994.
- [25] Sessions, R. **COM and DCOM : Microsoft's vision for distributed objects**. Wiley., 1998.
- [26] Sun Microsystems **Enterprise JavaBeans technology specification, version 2.1 – final release**. *Technical Report*., 1995.
- [27] Sun Microsystems **JavaBeans component architecture documnetation, Technical Report**., 1995.
- [28] De Win, B., Piessens, F., Smans, J. and Joosen, W. 2005. **Towards a unifying view on security contracts**. *Proceedings of the 2005 workshop on Software engineering for secure systems--building trustworthy applications - SESS '05* (St. Louis, Missouri, 2005), 1-7.