Evolution of reusable interfaces in product development

March 1st, 2013

Dr. Simon Giesecke, Dr. Niels Streekmann

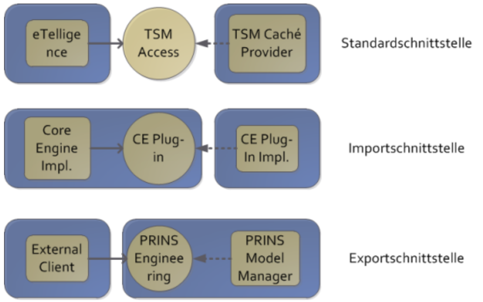
BTC Business Unit „Energy Products“

- Founded in January 2011
- Bundles efforts in product development
- Currently focus on products for the energy sector
  - EPM = **Energy Process Management**
- Product range
  - Established products, e.g. Grid Control System **PRINS**
  - Products evolved from individual project, e.g. **Wind Farm Center**
  - New products, e.g. **Advanced Meter Management**

  - The products are <u>customizable standard products</u>
    - i.e. not sold off-the-shelf, but customized in large projects by BTC or partners

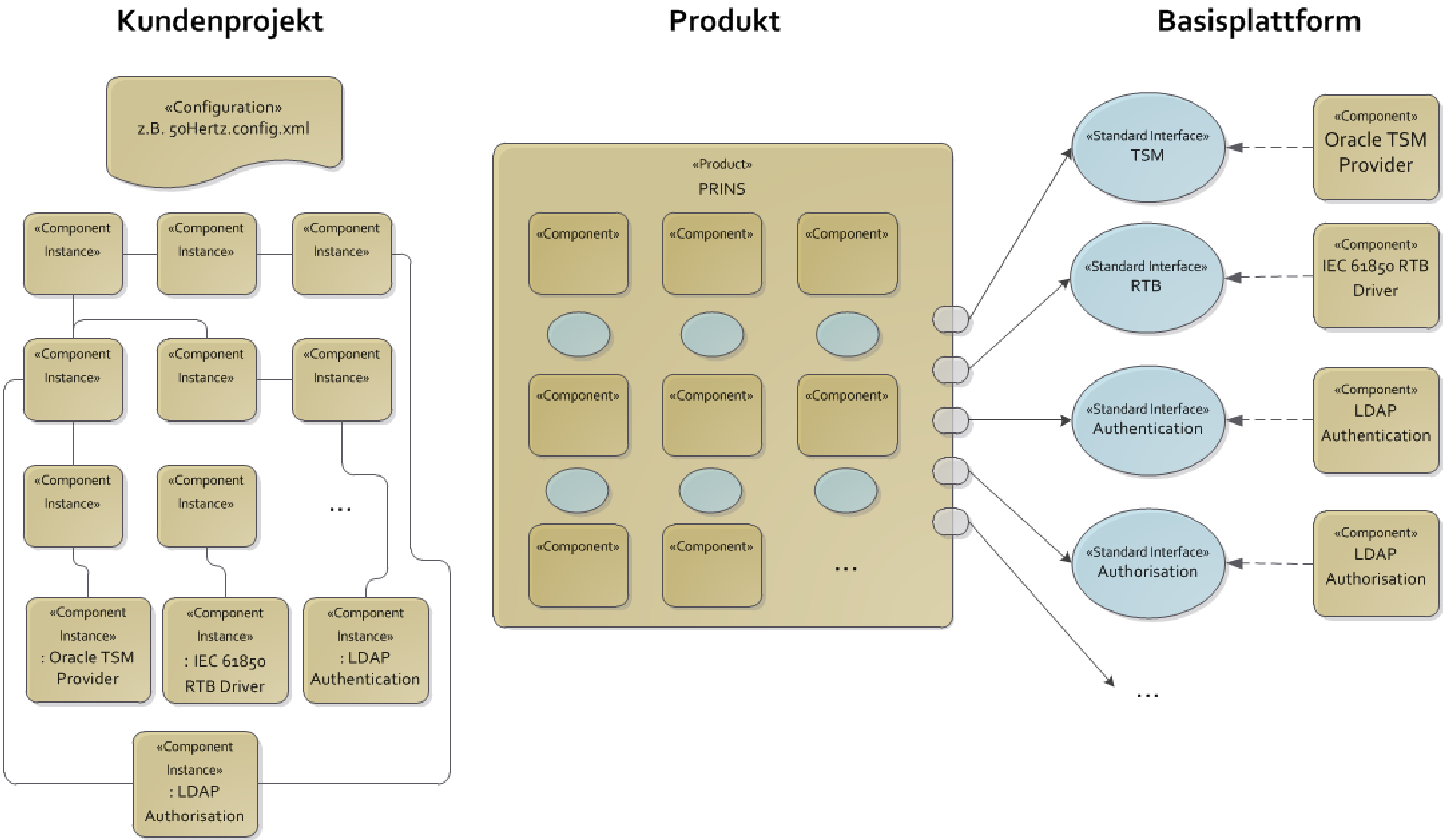
Characteristics of product development at BTC

- **Development mainly in C++/native and C#/.NET**
- **Different products developed on a common basic platform of framework libraries, standard interfaces and reusable components**
- **Delivery of modules as separate dynamic libraries**
- **Component-based approach: Separation of interfaces and implementations into separate modules**
- **Classification of interfaces into standard, export and import interfaces (Quasar)**



- **Production systems are hard to update**
  - availability requirements
  - complex engineering data that would need to be migrated
  - old versions may run for a long time (years)

Development model



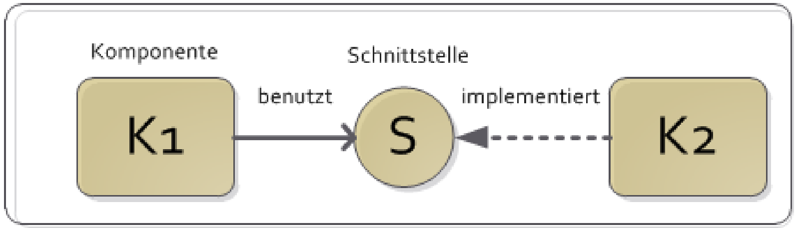
Evolution scenarios in product development

1. **Identification of an implementation bug in a component or a framework library after it has been deployed to a customer**

Best case:

- only one dynamic library must be replaced

2. **Request from product development team for a new feature in a basic platform component K2 that can only be handled by extending a standard interface S**



Best case:

- S is extended without affecting existing clients and implementations
- only K2 needs to implement the extended interface

Interface evolution

Basic guideline:

Compatibility should be guaranteed across releases.

- **Must maintain compatibility in minor and bug-fix revisions.**
- **May break compatibility in major revisions.**

Questions:

- **When to release the first version or major revision of an interface?**
- **What does compatibility mean? -> dimensions of compatibility**
- **How to check compatibility? -> tool support**

When to release the first version or major revision of an interface?

„Rele
relea
And lis
cus
(E
Ra

PRE-release early, PRE-release often.
(And listen to your customers.)
Ensure that PRE-releases never go into production.

tle
s
le“
r)

- **Not strictly true for APIs**
- **Compilers and processors are pretty non-adaptive fellows**

- **„as little as possible“ is not nothing**
- **„as late as possible“ is not never**
- **do not force clients to use non-published APIs**

When to release? – Some more thoughts

In our product development setting:

- We publish only very few interfaces to the outside.
- So the „customers“ for most interfaces are close.
- We have more than one chance to get it right, at least before software is deployed to an external customer.
- Incompatible changes to an interface are possible, but still, significant costs are associated with them.

Dimensions of compatibility

Syntactic vs. semantic compatibility

Source vs. binary compatibility

Strict vs. weak compatibility

Client vs. provider compatibility

Definition of binary compatibility

A change to a type is *binary compatible with* (equivalently, does not *break binary compatibility* with) pre-existing binaries

- **if the behaviour of pre-existing binaries (that use the type) is not affected**. (= strict binary compatibility)

Unfortunately, this excludes any “bug fix” and only allows strict extensions.

- **if pre-existing binaries that previously linked without error will continue to link without error**. (= weak binary compatibility)

(from Java Language Specification, section 13.2)

This does not imply anything on behaviour, e.g. that the result is executable without error or at all.

Binary compatibility does not imply equivalence of binary & source replacement!

Framework library:

```
struct X : Object {
  void Equals(const Object &other); //since V1.0

};
```

Client:

```
X a, b;
a.Equals(b);
```

compile against V1.0, dynamic link against V1.0

Binary compatibility does not imply equivalence of binary & source replacement!

Framework library:

```
struct X : Object {
  void Equals(const Object &other); //since V1.0
  void Equals(const X &other); //since V1.1
};
```

Overload resolution is performed at compile time (in Java, C#, C++)

Client:

```
X a, b;
a.Equals(b);
```

compile against V1.1, dynamic link against V1.1

compile against V1.0, dynamic link against V1.0 or V1.1

compile against V1.0, dynamic link against V1.0

How can compatibility be checked?

By tests.

But tests are often

… far from being complete.

… wrong, rely on implementation details, even bugs.

So this should be complemented (not replaced!) by other means.

Tool prototype: CSAPIDiff

Identifies syntactic API changes and classifies their consequences for binary/source client/provider compatibility.

Currently for C#/.NET assemblies (hence the name), but extensible.

Main use cases:

- **BEFORE release: Monitor/verify compatibility of changes before new minor/bug-fix release**
- **ON release: Support creation of release notes**
- **AFTER release: Support adaptation of client/provider code (in case of incomplete/non-existent release notes)**

Example: Spring.NET Spring.Core Breaking Changes 1.2.x->1.3.0 (1/2)

1. *within an ValidationGroup element (<v:group>,<v:exclusive>,..), n elements now must occur after any <v:message>, <v:action> or < elements. […]*
2. XmlReaderContext constructor now requires an IObjectDefinitionFactory to be specified. Thus **XmlReaderContext.ObjectDefinitionFactory is read only now**.
3. Changes to the Apache NMS API, which was not yet a public release when included in Spring 1.2.0 made breaking API changes. On NmsTemlate,
   1) The property 'byte Priority' was changed to 'MsgPriority Priority'
   2) The property 'bool Persistent' is no longer part of the NMS API but is still supported in a backward compatible manner by Spring by translation to standard MsgDeliveryMode enumeration values of Persistent and NonPersistent. A new property MsgDelivery has been added. The class, CachedMessageProducer, which is unlikely to be use by end users, was directly upgraded to the la
   any backwards compatibility support.

This is not an API, but Data Format change.

This is not actually in Spring.Core.dll, but in Spring.Nms.dll

Example: Spring.NET Spring.Core Breaking Changes 1.2.x->1.3.0 (2/2)

4. **AbstractApplicationContext.CaseSensitive renamed to AbstractApplicationContext.IsCaseSensitive**

> Only relevant to providers.

5. Spring.Validation: Base class BaseValidator changed to BaseSimpleValidator for single validators as compared to group validators *which now comi*
BaseGroupValidator instead of ValidatorGroup

> Not yet checked

6. **IVariableSource** implementations now must also implement **CanResolveVariable(string** variableName) and m*ay throw exceptions in*
ResolveVariable() in case the variable cannot resolved by this p
source. In order to distinguish between an existing variable hav
a non-existing variable, variable sources need to be changed to

> Only relevant to providers; exceptions not declared in .NET

7. a) Signature of **CollectionUtils.Contains(ICollection** collection,
changed to CollectionUtils.Contains(**IEnumerable** collection, obj
b) returns 'null' in case of collection==null instead of throwing an

> b) behavioural change

8. **dropped Spring.Expressions.DateLiteralNode**

(from http://www.springframework.net/docs/1.3.0/BreakingChanges.txt)

Example CSAPIDiff output

Analysis set

Analysis unit	Version in before model	Version in after model
Spring.Core	1.2.0.20001	1.3.0.20001

API Changes

Subject	Message	Client source compatible	Client binary compatible	Provider source compatible	Provider binary compatible
Spring.Context.Support.AbstractApplicationContext. Void ProcessObjectFactoryPostProcessors(System.Collections.IList)	Parameter orderedFactoryProcessors renamed to objectFactoryPostProcessors	No	Yes	No	No
Spring.Expressions	Type Spring.Expressions.BaseNode+EvaluationContext removed	No	No	Maybe	Maybe
Spring.Expressions	Type Spring.Expressions.DateLiteralNode removed	No	No	Maybe	Maybe
Spring.Expressions.LambdaExpressionNode	Public member GetValueInternal removed	No	No	Maybe	Maybe
Spring.Expressions.SpringAST. Void initialize(antlr.collections.AST)	Parameter type of parameter t changed from antlr.collections.AST to Spring.Expressions.Parser.antlr.collections.AST	No	No	No	No
Spring.Expressions.SpringAST. Void initialize(antlr.IToken)	Parameter type of parameter tok changed from antlr.IToken to Spring.Expressions.Parser.antlr.IToken	No	No	No	No
Spring.Objects.Factory.Config.ObjectDefinitionVisitor. System.Object ResolveStringValue(System.String)	Return type specialized from System.Object to System.String	Yes	No	No	No
Spring.Objects.Factory.Config.ObjectDefinitionVisitor. System.Object ResolveStringValue(System.String)	Parameter variableName renamed to rawStringValue	No	Yes	No	No
Spring.Objects.Factory.Config.ObjectDefinitionVisitor. Void VisitManagedDictionary(Spring.Objects.Factory.Support.ManagedDictionary)	Parameter type of parameter dictVal changed from Spring.Objects.Factory.Support.ManagedDictionary to Spring.Objects.Factory.Config.ManagedDictionary	No	No	No	No
Spring.Objects.Factory.Config.ObjectDefinitionVisitor. Void VisitManagedList(Spring.Objects.Factory.Support.ManagedList)	Parameter type of parameter listVal changed from Spring.Objects.Factory.Support.ManagedList to Spring.Objects.Factory.Config.ManagedList	No	No	No	No

8

What software developers should know by heart

- **The SOLID principles:**
  - Single responsibility principle
  - Open/closed principle
  - Liskov substitution principle
  - Interface segregation principle
  - Dependency inversion principle
- **Design-by-contract**
- **Encapsulation of implementation details**
- **Run-time vs. compile-time polymorphism**
- **Refactoring vs. changing published interfaces**

Research challenges / technical

- Since syntactic compatibility can be automatically checked:

  Do semantic API changes coincide with syntactic API changes?
  Under what circumstances is this not the case?
  Can the relationship be characterized in more detail?
- How can coexistence of multiple module versions within an application be exploited best?

Many technical questions can be and have been explored in practise, but we need facts on the economic consequences.

Research challenges / economic

Under incomplete knowledge & uncertainty, improve on gut-feelings and opinion by providing hard data:

- Which way of extension is the most cost-efficient?
- When should standard interfaces be used, when should one stick with export interfaces?
- When to invest in extensibility?
- When is breaking compatibility more economic than maintaining compatibility?

This may build on work on “The structure and value of modularity in software design” (Sullivan/Griswold/Cai/Hallen)

General: Theory that reliably allows the prediction of the impact of any design decision on maintenance costs.

A final note: C++ vs. Java/C#

The situation is more complex in C++, and tool support is much weaker.

But: „The world is built on C++“ (Herb Sutter)

Additional challenges in C++ make binary compatiblity hard:

- **Compiler-dependent ABI**
- **Prevalence of in-header code (inline methods, template methods) and compile-time inlining**
- **Compile-time fixation of memory layout**
- **Optimization of virtual method calls into non-virtual method calls**
- **Implicit type conversions**
- **…**

Good news: no reflection in C++

Sources on compatibility in practice

- Focus on Java: http://wiki.eclipse.org/index.php/Evolving_Java-based_APIs
- Focus on C++: http://techbase.kde.org/Policies/Binary_Compatibility_Issues_With_C%2B%2B

BTC AG Offices

Headquarter:

Escherweg 5
26121 Oldenburg
Fon: + 49 441 3612-0
Fax: + 49 441 3612-3999
E-Mail: office-ol@btc-ag.com
www.btc-ag.com

Kurfürstendamm 33
10719 Berlin
Fon: + 49 30 88096-5
Fax: + 49 30 88096-777
E-Mail: office-b@btc-ag.com

Weser Tower
Am Weser-Terminal 1
28217 Bremen
Fon: +49 421 33039-0
Fax: +49 421 33039-399
E-Mail: office-hb@btc-ag.com

Bartholomäusweg 32
33334 Gütersloh
Fon: +49 5241 9463-0
Fax: +49 5241 9463-55
E-Mail: office-gt@btc-ag.com

Burchardstraße 24
20095 Hamburg
Fon: +49 40 210098-0
Fax: +49 40 210098-76
E-Mail: office-hh@btc-ag.com

Klostergasse 5
04109 Leipzig
Fon: +49 341 350558-0
Fax: +49 341 350558-59
E-Mail: office-l@btc-ag.com

Wilh.-Th.-Römheld-Str. 24
55130 Mainz
Fon: + 49 6131 88087-0
Fax: + 49 6131 88087-99
E-Mail: office-mz@btc-ag.com

Türkenstraße 55
80799 München
Fon: +49 89 3603539-0
Fax: +49 89 3603539-59
E-Mail: office-m@btc-ag.com

An der Alten Ziegelei 1
48157 Münster
Fon: +49 251 14132-0
Fax: +49 251 14132-11
E-Mail: office-ms@btc-ag.com

Konrad-Zuse-Straße 3
74172 Neckarsulm
Fon: +49 (7132 380-0
Fax: +49 7132 380-29
E-Mail: office-nsu@btc-ag.com

Çayiryolu 1, Partaş Center Kat: 11-12
Içerenköy, 34752 Istanbul
Türkei
Fon: +90 (216) 5754590
Fax: +90 (216) 5754595
E-Mail: office-ist@btc-ag.com

ul. Małe Garbary 9
61-756 Poznań
Polen
Fon: +48 (0) 61 8560970
Fax: +48 (0) 61 8501870
E-Mail: biuro-poz@btc-ag.com

Hasebe Build.11F,
4-22-3 Sendagi, Bunkyo-Ku,
113-0022 Tokyo
Japan
Fon: +81 (3) 5832 7020
Fax: +81 (3) 5832 7021
Email: Info.OSCJapan@btc-es.de

Bäulerstraße 20
CH-8152 Glattbrugg
Schweiz
Fon: +41 (0) 44 874 3000
Fax: +41 (0) 44 874 3010
E-Mail: office-zh@btc-ag.com

Thank you very much for your attention.

BTC Business Technology Consulting AG
Escherweg 5
26121 Oldenburg
Ph. 0441 / 36 12-0
www.btc-ag.com

simon.giesecke@btc-ag.com