

Integration Testing of Object-Oriented .NET Applications

Dr. Andreas Ulrich, Dr. Jan Wieghardt, Dr. Andrej Pietschker

Siemens AG, Corporate Technology
 {andreas.ulrich, jan.wieghardt}@siemens.com

Giesecke & Devrient GmbH, Banknote Processing
 andrej.pietschker@gi-de.com

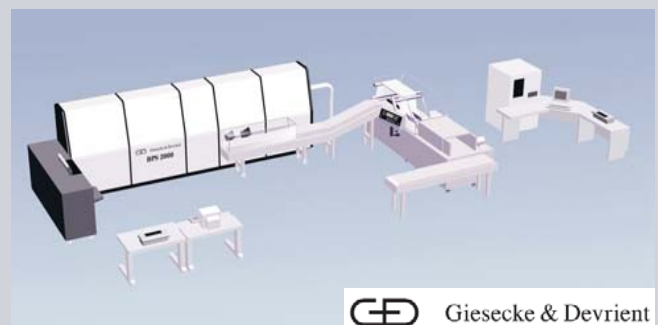
Copyright © Siemens AG 2008. All rights reserved.

Introduction

BPS 3000

These units are deployed at central banks around the world for

- Counting,
- Inspection,
- Sorting,
- Packing,
- Destruction of banknotes.



Task

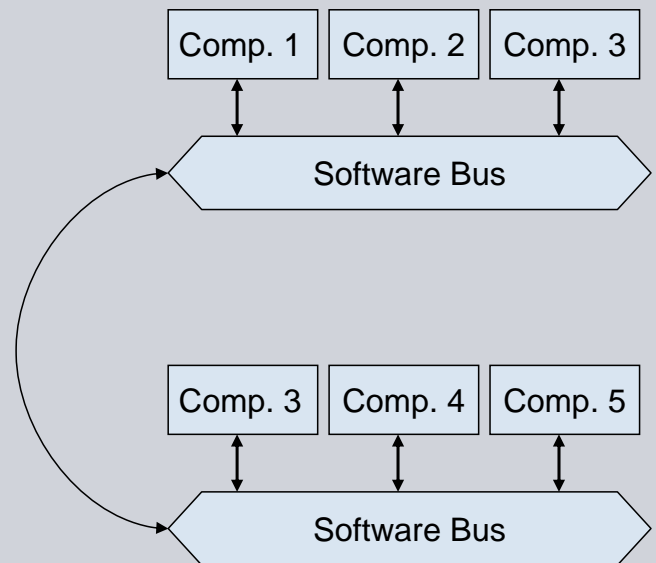
A test approach and test architecture for the system's next generation central control application software was to be devised.



The Control Application

Overview

- Implemented as C# .net application
- Based on a component framework
- Components communicate via a software bus exchanging C# objects
- The software bus implements a publisher subscriber pattern
- Different instances of the bus may be linked across process and machine boundaries via .net remoting
- Potentially very complex C# objects
- New C# communication objects can be introduced by each component – currently more than 500 different object types.



Test Architecture

Test Goal

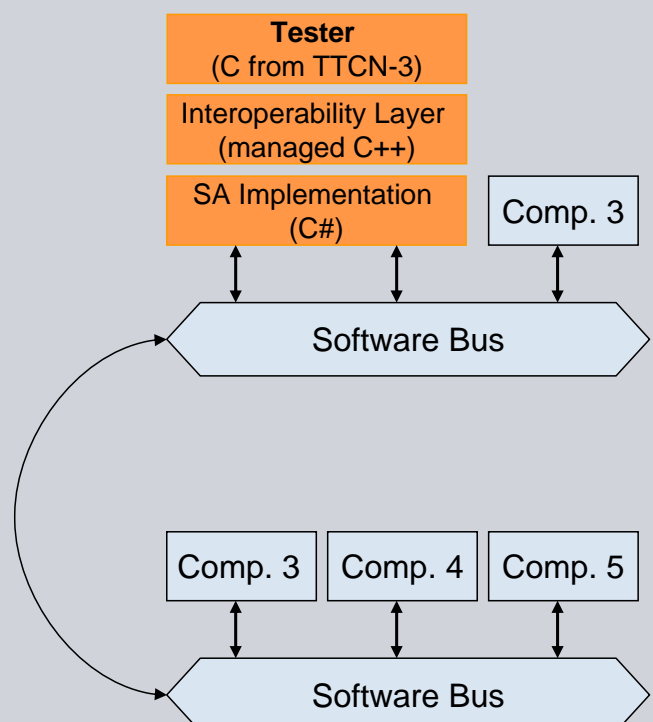
- Integration testing of any configurable sub-system
- Use of a TTCN-3-to-C compiler

Overview

- Components not part of the system under test (SUT) are replaced by the tester.
- The system is stimulated and observed by sending and receiving the C# objects

Challenges

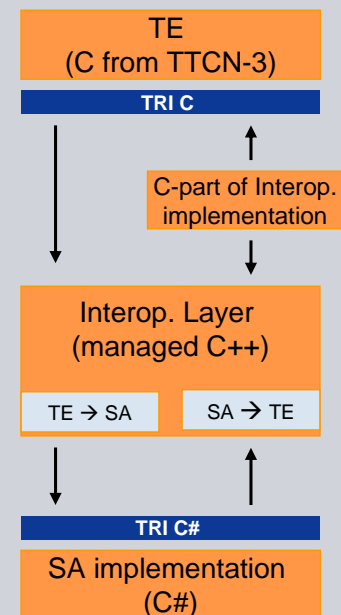
- C/C# interoperability between tester and SUT
- C# object type representation in TTCN3 (for over 500 messages)
- Transfer between TTCN-3 messages and C# object representations



C/C# Interoperability

Solution

- Calls to the system adaptor (SA) are redirected to an interoperability layer.
- The interoperability layer is written in managed C++ and exposes a C-style DLL interface and makes calls to a .net TRI-façade.
- The façade is again implemented in C# and provides access to an interface providing the TRI equivalent in C#.
- The marshalling from C to .net types is realized in the interop layer and is based on the .net support for interoperability.
- The TRI design in .net is straight forward and was inspired by the Java language mapping described by the TTCN-3 standard.
- Calls from the SA to the TE are stored in queues within the interop layer which are monitored within in a separate thread of the C implementation.
- The SA implementation may be in any .net language.



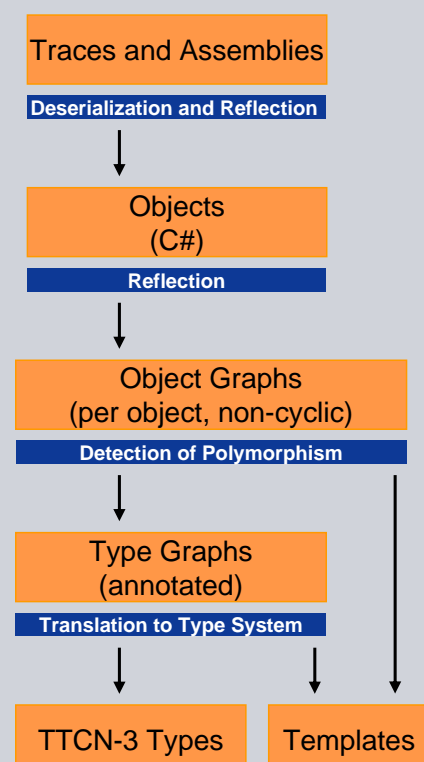
C# Object Type Representation in TTCN-3

Challenge

- Reach consistent data type representations between C# and TTCN-3

Solution

- Based on serializable C# type definitions
- Use .net reflection and serialization to construct instances of C# types automatically from assemblies or traces of recorded C# objects.
- Traverse and analyze object graph using reflection of each object.
- Detect polymorphism (ref to different types) and possibility to `omit` (ref to null).
- Create type graph from object graphs.
- Dump TTCN-3 representation of type graphs.
- Dump TTCN-3 send/receive template prototypes



C# Object Type Representation in TTCN-3 (Examples)

Assemblies to Groups

```
/* mscorlib, Version=1.0.5000.0, Culture=neutral*/
group mscorlib{
  type record System_DateTime{
    System_Int64 ticks
  };

  type record of System_Byte System_Byte_ARRAY;
}
with {
  variant "mscorlib, Version=1.0.5000.0,[...]"
};
```

represents System.DateTime

```
class DateTime {
  private int64 ticks;
}
```

represents the build-in type array
System.byte[]

Polymorphism

```
type union Nmspc_MyShape{
  Nmspc_Triangle variant_Nmspc_Triangle;
  Nmspc_Square variant_Nmspc_Square;
}

type record Nmspc_MyShapeContainer{
  Nmspc_MyShape theShape;
}
```

```
namespace Nmspc {
  class MyShape { ... };

  class Triangle : MyShape { ... };
  class Square : MyShape { ... };

  class MyShapeContainer{
    private Shape theShape;
  } }
```

TCCN-3 Message to C# Object Representation

Challenge

- Bijective mapping between C# and TTCN-3 object representations.

Solution

- The codec implements a **transfer syntax** that allows to serialize TTCN-3 messages to a byte array and in turn deserialize C# objects from the byte array and vice versa.
- Each element of a C# object is encoded in terms of its *name* and *type* including *namespace* and *assembly*.
- *Name*, *type*, and *assembly* uniquely identify and instantiate the TTCN-3 and C# representations of messages and objects.

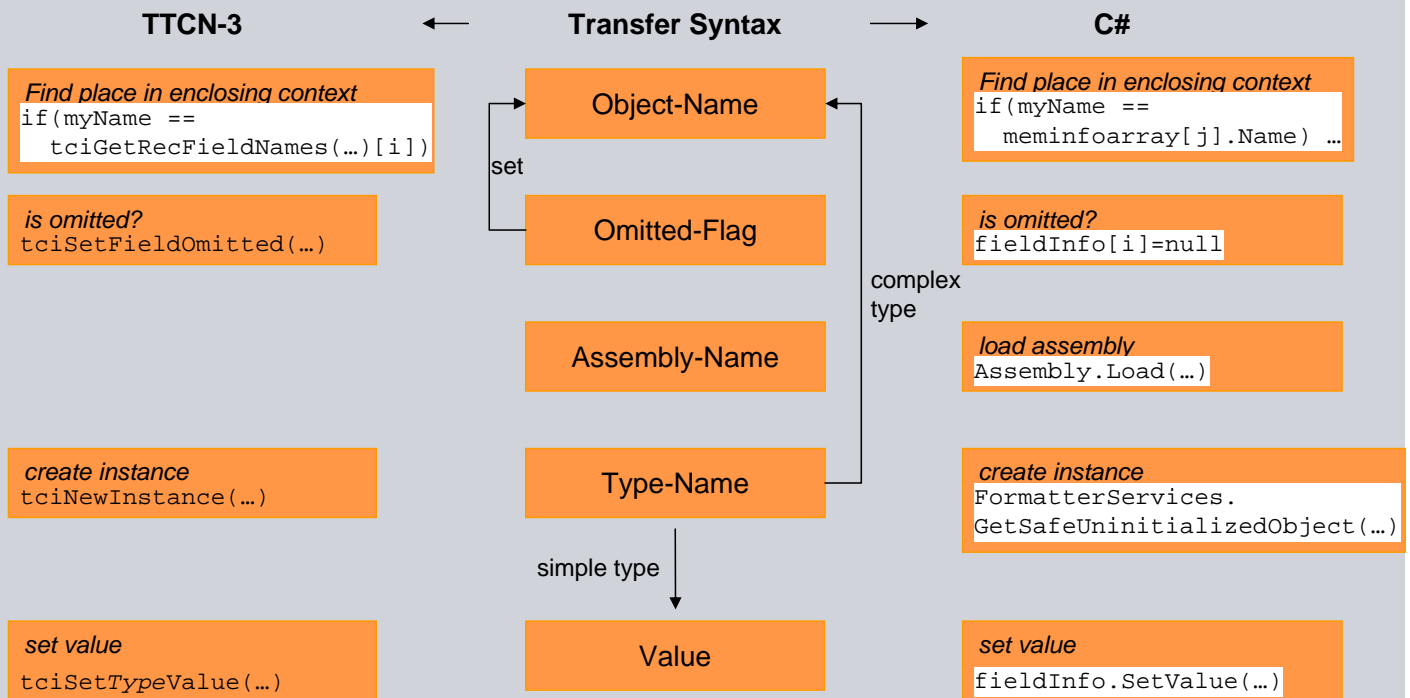
Limits

- C# objects must be serializable.
- A C# object graph must resemble a tree, i.e. cycle-free and no multiple references to single objects.

→ Changing the transfer syntax, limitations above could be overcome.

→ A more elaborated approach was not necessary in the context of our project.

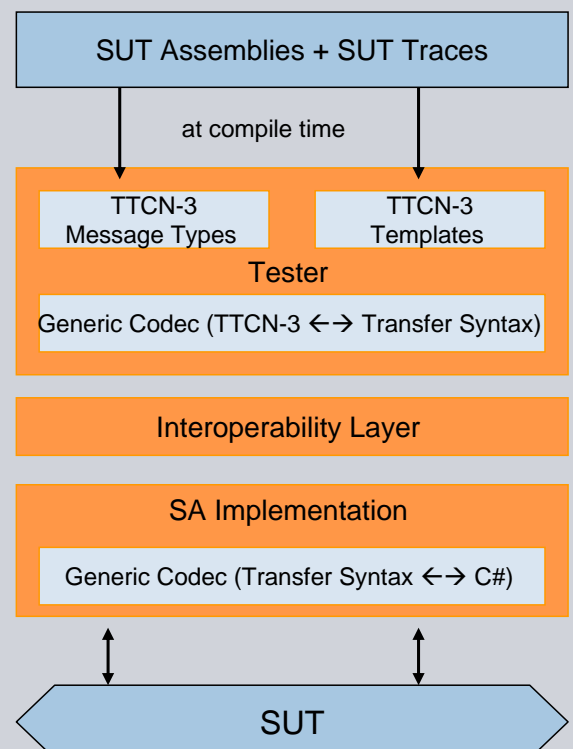
Decoding of Transfer Syntax to TTCN-3 and C#



Test Architecture (Revisited)

Main Features

- Fully automatic generation of the TTCN-3 type system from C# guarantees coherent mapping of relevant types.
- The codec is applicable to a large subset of C# objects.
- The generality of the codec and the automatic generation allows to handle a large and varying set of C# communication objects.
- Template prototypes facilitate test case implementation, which is desirable due to complexity of the C# object types.



Conclusions

Starting from a very specific project it was possible and necessary to arrive at a generic solution relevant to the general problem of testing .net applications.

- The TRI C language mapping is mapped to a .net language representation. The approach is straightforward and no restrictions apply.
- A representation of C# types in TTCN-3 can be generated from assemblies and recorded execution traces (serialized C# objects).
 - Objects need to be serializable. This precondition is often met in message orientated .net frameworks with remoting capabilities.
 - Solutions were found to represent certain aspects of polymorphism in TTCN-3.
 - Approach is sufficiently generic to handle also TTCN-3 external function calls to .net.
- A codec and a transfer syntax were designed to be independent from the TTCN-3 or C# type representation of the communication objects.
 - Type changes do not influence the codec implementation.
 - The current restriction for cyclic object graphs can be overcome if necessary.