

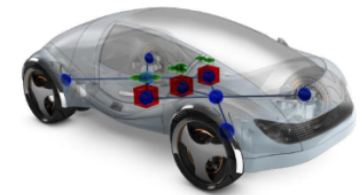
Joachim Fröhlich, Sylvia Jell and Andreas Ulrich

# Applying TDL to describe tests of a distributed RT control system

# Robust and reliant automotive computing environment for future eCars

## RACE: Project objectives

- Platform of distributed, redundant nodes for fail-operational functions
- Centralized platform core reduces complexity and requires less control units
- More functionality realized in software
- Plug & play for new features, components, nodes, e.g. Adaptive Cruise Control
- Improved approval capability of the ICT architecture



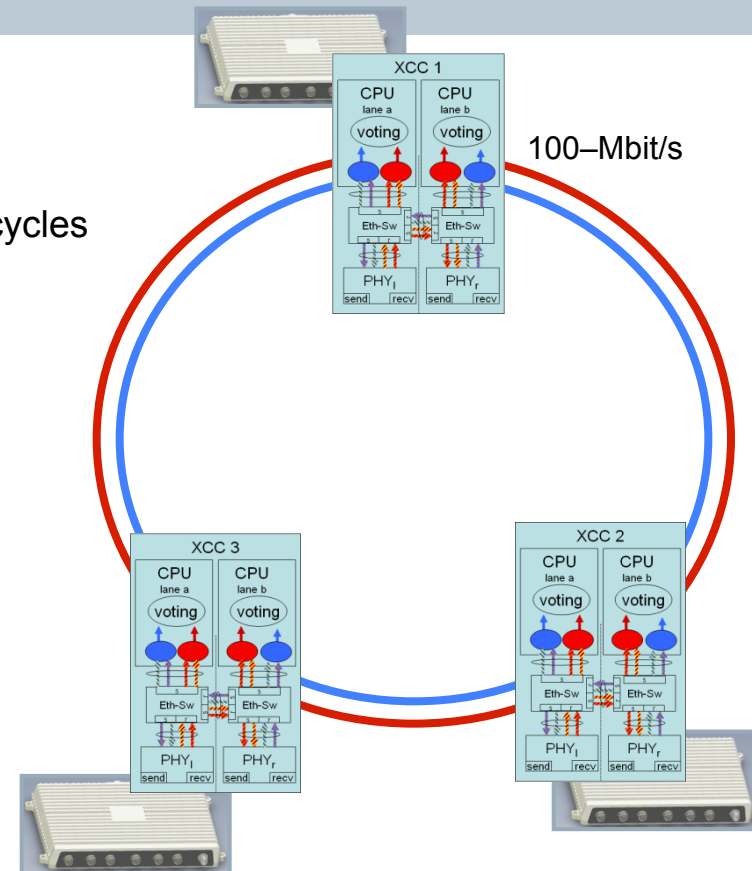
**Test of safety services in hard-real-time without side-effects, early on in development**



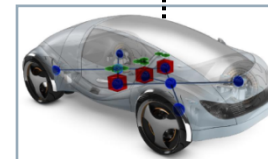
# Approach: Distributed system of redundant, testable nodes running time-triggered

## Time-triggered execution of nodes of the RACE platform

- Platform core of several XCCs of 2 redundant lanes (X = 2)
- Platform services and applications run periodically in 10ms cycles
- Data flows between component functions in defined order
- 100 Mbit/s Ethernet for communication, synchronized clocks
- Use of synchronous dataflow programming and testing



Vehicle Control Computer (VCC)



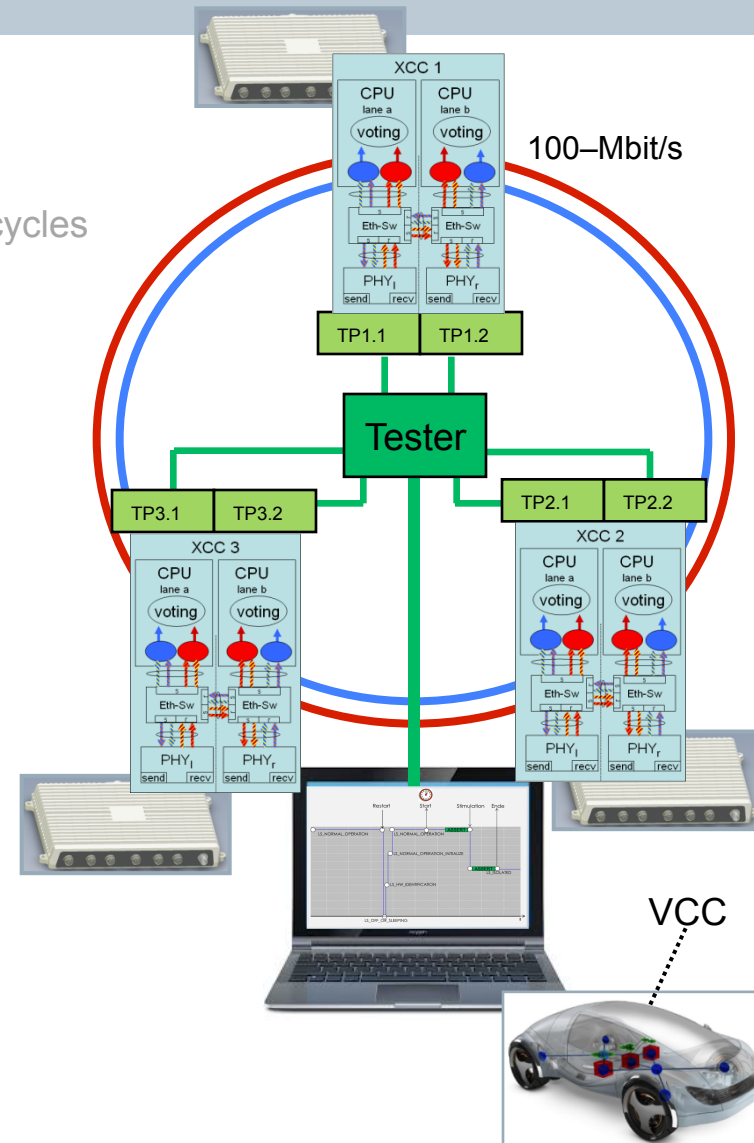
# Approach: Distributed system of redundant, testable nodes running time-triggered

## Time-triggered execution of nodes of the RACE platform

- Platform core of several XCCs of 2 redundant lanes ( $X = 2$ )
- Platform services and applications run periodically in 10ms cycles
- Data flows between component functions in defined order
- 100 Mbit/s Ethernet for communication, synchronized clocks
- Use of synchronous dataflow programming and testing

## Test probes (SW/HW) seamlessly built-in in each node

- Run in an reserved time-slot at the end of each cycle
- Observe states and signals (SUT output)
- Manipulate states and signals (test stimulus)
- Seed data and inject faults (test stimulus)
- Common controlling and coordinating tester: RT-Linux
- Dedicated, fast p2p communication links (Eth)
- Interactive test HMI on separate machine



# How to specify and document tests?

## The need for a test description language

- Specification in a programming language, e.g. C?
  - (+) Directly executable
  - (-) Lost in details; what shall be tested?

```
unsigned int __fastcall sub_10002(int a1)
{
    signed int v1; // ecx@1
    int v2; // edi@1
    bool v4; // zf@3

    v2 = a1;
    v1 = -1;
    do
    {
        if ( !v1 )
            break;
        v4 = *(BYTE *)v2++ == 0;
        --v1;
    }
    while ( v4 );
    return ~v1;
}
```

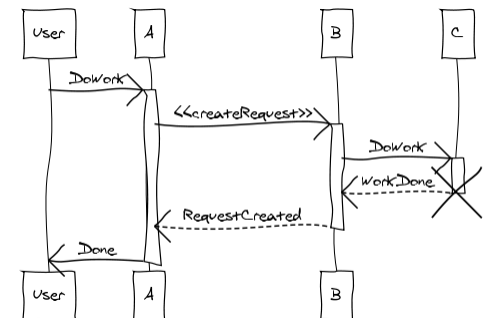
# How to specify and document tests?

## The need for a test description language

- Specification in a programming language, e.g. C?
  - (+) Directly executable
  - (-) Lost in details; what shall be tested?
- Specification in a graphical language, e.g. standard UML?
  - (+) Good to capture high-level overview
  - (-) Various semantics, tool dependent; mostly considered as artwork

```
unsigned int __fastcall sub_10002(int a1)
{
    signed int v1; // ecx@1
    int v2; // edi@1
    bool v4; // zf@3

    v2 = a1;
    v1 = -1;
    do
    {
        if ( !v1 )
            break;
        v4 = *(BYTE *)v2++ == 0;
        --v1;
    }
    while ( v4 );
    return ~v1;
}
```



# How to specify and document tests?

## The need for a test description language

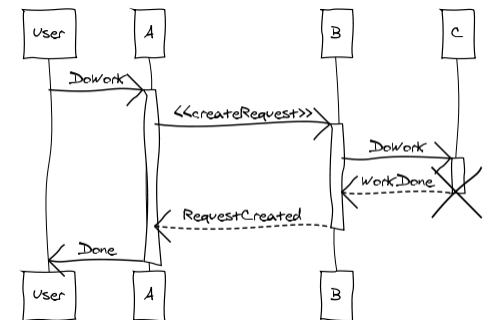
- Specification in a programming language, e.g. C?
  - (+) Directly executable
  - (-) Lost in details; what shall be tested?
- Specification in a graphical language, e.g. standard UML?
  - (+) Good to capture high-level overview
  - (-) Various semantics, tool dependent; mostly considered as artwork
- Reality: Mostly plain text; use of MS Word or Excel or Adobe PDF

```

unsigned int __fastcall sub_10002(int a1)
{
    signed int v1; // ecx@1
    int v2; // edi@1
    bool v4; // zf@3

    v2 = a1;
    v1 = -1;
    do
    {
        if ( !v1 )
            break;
        v4 = *(BYTE *)v2++ == 0;
        --v1;
    }
    while ( v4 );
    return ~v1;
}

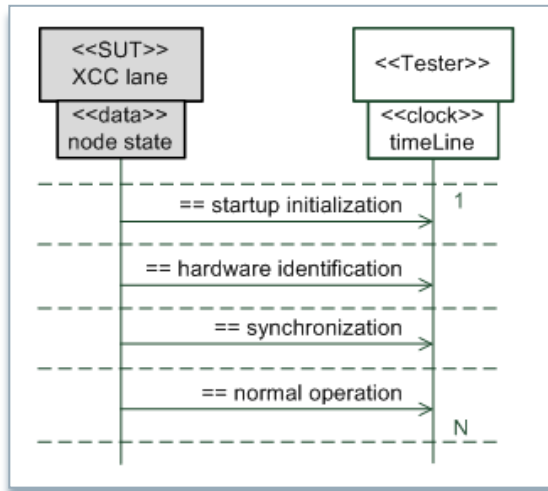
```





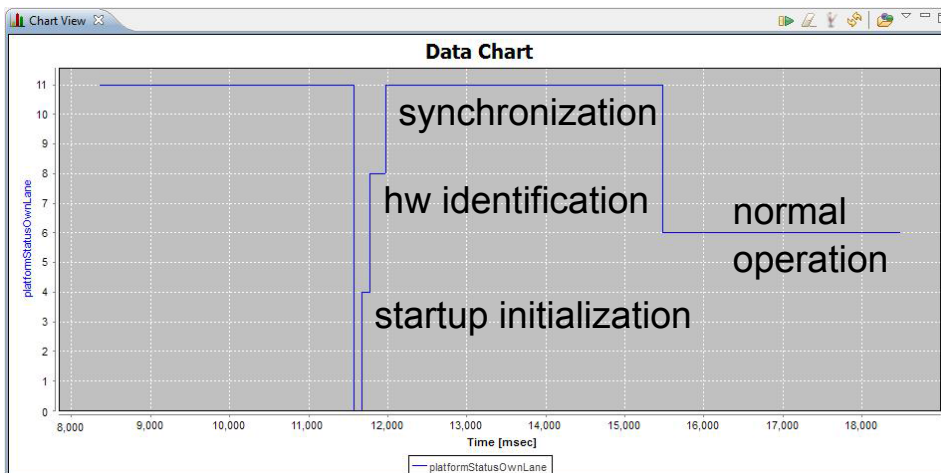
# Test descriptions in RACE – Expression of domain concepts by extending UML sequence diagrams

## Example test “XCC lane boots up in max. N cycles”



```

Terminal - race@fiero: ~/rte-src/build/multiProcessAuto_Linux_Rele...
File Edit View Terminal Go Help
race@fiero: ~/rte-src
target/run_dcc1A --test=1,101 | grep TstPrb
TstPrb : cycle @1: Port[9.0.0 = 589824] = 1
TstPrb : cycle @2: Port[9.0.0 = 589824] = 9
TstPrb : cycle @3: Port[9.0.0 = 589824] = 10
TstPrb : cycle @4: Port[9.0.0 = 589824] = 10
TstPrb : cycle @5: Port[9.0.0 = 589824] = 12
TstPrb : cycle @6: Port[9.0.0 = 589824] = 12
TstPrb : cycle @7: Port[9.0.0 = 589824] = 13
TstPrb : cycle @8: Port[9.0.0 = 589824] = 13
TstPrb : cycle @1: Port[9.0.0 = 589824] = 14
TstPrb : cycle @2: Port[9.0.0 = 589824] = 14
TstPrb : cycle @3: Port[9.0.0 = 589824] = 11
TstPrb : cycle @4: Port[9.0.0 = 589824] = 2
TstPrb : cycle @5: Port[9.0.0 = 589824] = 16
TstPrb : cycle @6: Port[9.0.0 = 589824] = 3
TstPrb : cycle @7: Port[9.0.0 = 589824] = 3
TstPrb : cycle @8: Port[9.0.0 = 589824] = 3
TstPrb : cycle @9: Port[9.0.0 = 589824] = 3
TstPrb : cycle @10: Port[9.0.0 = 589824] = 3
TstPrb : cycle @11: Port[9.0.0 = 589824] = 3
TstPrb : cycle @12: Port[9.0.0 = 589824] = 3
TstPrb : cycle @13: Port[9.0.0 = 589824] = 3
TstPrb : cycle @14: Port[9.0.0 = 589824] = 3
TstPrb : cycle @15: Port[9.0.0 = 589824] = 3
TstPrb : cycle @16: Port[9.0.0 = 589824] = 3
TstPrb : cycle @17: Port[9.0.0 = 589824] = 3
TstPrb : cycle @18: Port[9.0.0 = 589824] = 3
TstPrb : cycle @19: Port[9.0.0 = 589824] = 3
TstPrb : cycle @20: Port[9.0.0 = 589824] = 3
TstPrb : cycle @21: Port[9.0.0 = 589824] = 3
TstPrb : cycle @22: Port[9.0.0 = 589824] = 3
TstPrb : cycle @23: Port[9.0.0 = 589824] = 3
TstPrb : cycle @24: Port[9.0.0 = 589824] = 3
TstPrb : cycle @25: Port[9.0.0 = 589824] = 3
TstPrb : cycle @26: Port[9.0.0 = 589824] = 3
TstPrb : cycle @27: Port[9.0.0 = 589824] = 3
  
```



- 1) Periodic execution in cycles
- 2) 9.0.0 addresses a node's state variable
- 3) State 3 denotes "normal operation"



# Progressing beyond illustrative UML specifications

## The standardized ETSI Test Description Language (TDL)

### TDL offers

- Separation of test specification from test implementation
- Single, concise and comprehensive language on testing
- Support of black-box testing in different application domains
- Adjustable to stakeholders; multiple syntaxes

TDL abstract syntax, ES 203119-1  
TDL graphical syntax, ES 203119-2  
TDL exchange format, ES 203119-3

# Progressing beyond illustrative UML specifications

## The standardized ETSI Test Description Language (TDL)

### TDL offers

- Separation of test specification from test implementation
- Single, concise and comprehensive language on testing
- Support of black-box testing in different application domains
- Adjustable to stakeholders; multiple syntaxes

TDL abstract syntax, ES 203119-1  
TDL graphical syntax, ES 203119-2  
TDL exchange format, ES 203119-3

## TDL as the latest evolutionary step in test automation

**Test language**, capturing all test concepts in one language

**Keyword-driven testing**, keywords embedded in (natural) language

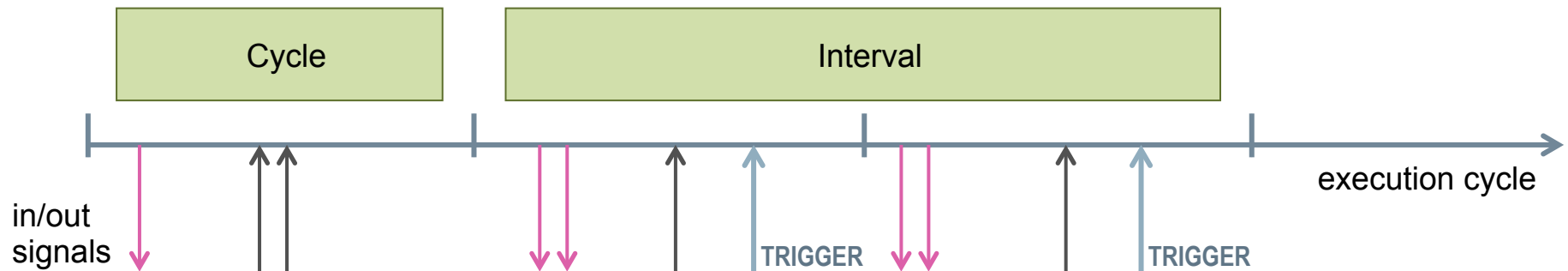
**Data-driven testing**, separation of test scripts and data

**Functional decomposition**, test frameworks

**Test automation**

# Finding abstractions to capture domain concepts is key for designing a good test description language

## Abstractions for the time-triggered system in RACE and mapping to TDL concepts



- In/out **signals** of the SUT == data exchanged with the environment → TDL interactions
  - **Trigger** signal == an expected signal is eventually output
- **Cycle** == behavior, in which signals are processed (in any order) → TDL compound behavior
  - **Interval** == behavior, which covers more than one cycle, bound to trigger signals
- **Step** == structural element of a test description, contains cycles, intervals
- Not all abstractions in RACE have fitting TDL representations → use of TDL annotations

# Building the RACE test language from TDL

## Tooling for TDL – From abstract syntax to concrete syntax

- EMFText Eclipse Plug-In (<http://www.emftext.org>)
- Starting point: existing TDL abstract syntax (from ETSI website)
- Generation of a first concrete syntax (in EBNF) from the TDL abstract syntax using EMFText
- Adaptation of generated concrete syntax according to domain requirements
  - Deletion of unneeded features
  - Introduction of domain concepts as keywords  
e.g. RACE signal → TDL interaction, RACE cycle → TDL compound behavior
  - Further simplification of syntax, optimization
    - Decide about the general language design  
e.g. use special symbols such as {, }, ;
- EMFText supports the generation of an **Eclipse-integrated editor**
  - Fully fledged editor with syntax-highlighting, code completion etc.



```

1 SYNTAXDEF tdl_mbat
2 FOR <http://www.etsi.org/spec/TDL/20130606>
3 START Package
4
5 IMPORTS {
6   types:<http://www.eclipse.org/uml2/5.0.0/Types>
7 }
8
9 RULES {
10
11 Package ::= "Package:" ("name" name['"', '"'] ";")?
12           (comment)* (annotation)*
13           ("owningPackage" owningPackage[] ";")?
14           (import)* (packagedElements)* ;
15
16 TestDescription ::= "TestDescription:" ("name" name['"', '"'] ";")?
17                  (comment)* (annotation)*
18                  ("owningPackage" owningPackage[] ";")?
19                  ("testObjective" testObjective[] ";")*
20                  (formalParameter)* !1
21                  "testConfiguration" testConfiguration[] ";"
22                  (behaviourDescription)? ;
23
24 TestObjective ::= "TestObjective:" ("name" name['"', '"'] ";")?

```

# The textual RACE test language derived from TDL and supported by an Eclipse built-in editor

## Example test “Continue execution after T2 stop cycles”

```

TestDescription:
  name "Test.1.d";
  Annotation: value "Node is in state S1", key Precondition, annotatedElement;
  owningPackage RaceTestSuite;
  testObjective Test.1.d;
  Parameter: ValidNodeState "S1";
  Parameter: Integer "T2";
  testConfiguration Config_1d;
  BehaviourDescription:
    STEP:
      STEP:
        TestDescriptionReference: Test.1.d.Precondition;

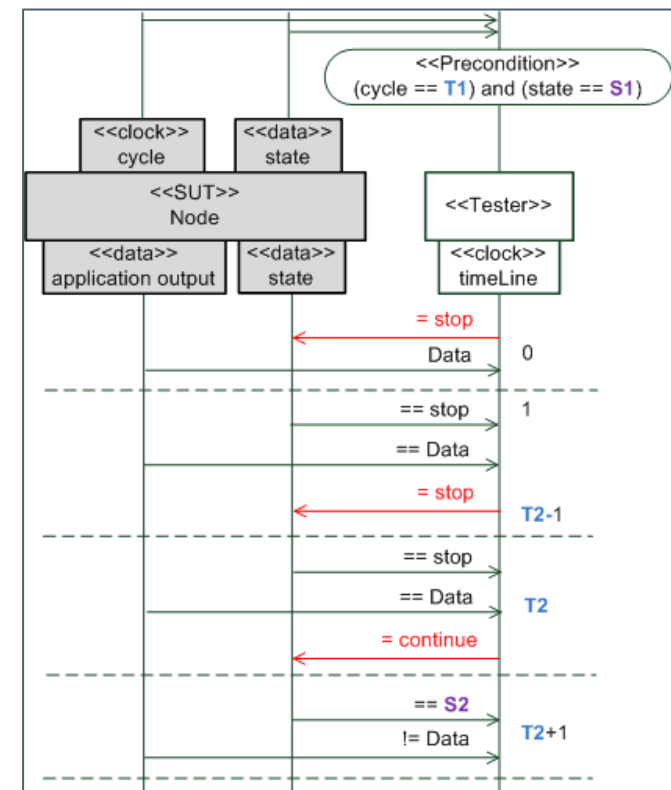
      STEP:
        Cycle
          signal from t to state : stop;
          signal from output to t : var_Data = any anyData;

      STEP:
        BoundedLoop: ( startValue: 1; endValue: add(var T2, -1); )
        Cycle
          signal from state to t : stop;
          signal from output to t : var var_Data;
          signal from t to state : stop;

      STEP:
        Cycle
          signal from t to state : stop;

```

## Graphical illustration of the test



# Conclusions – TDL enables design of domain-specific test languages

## TDL is a tool-independent approach to provide means for testing

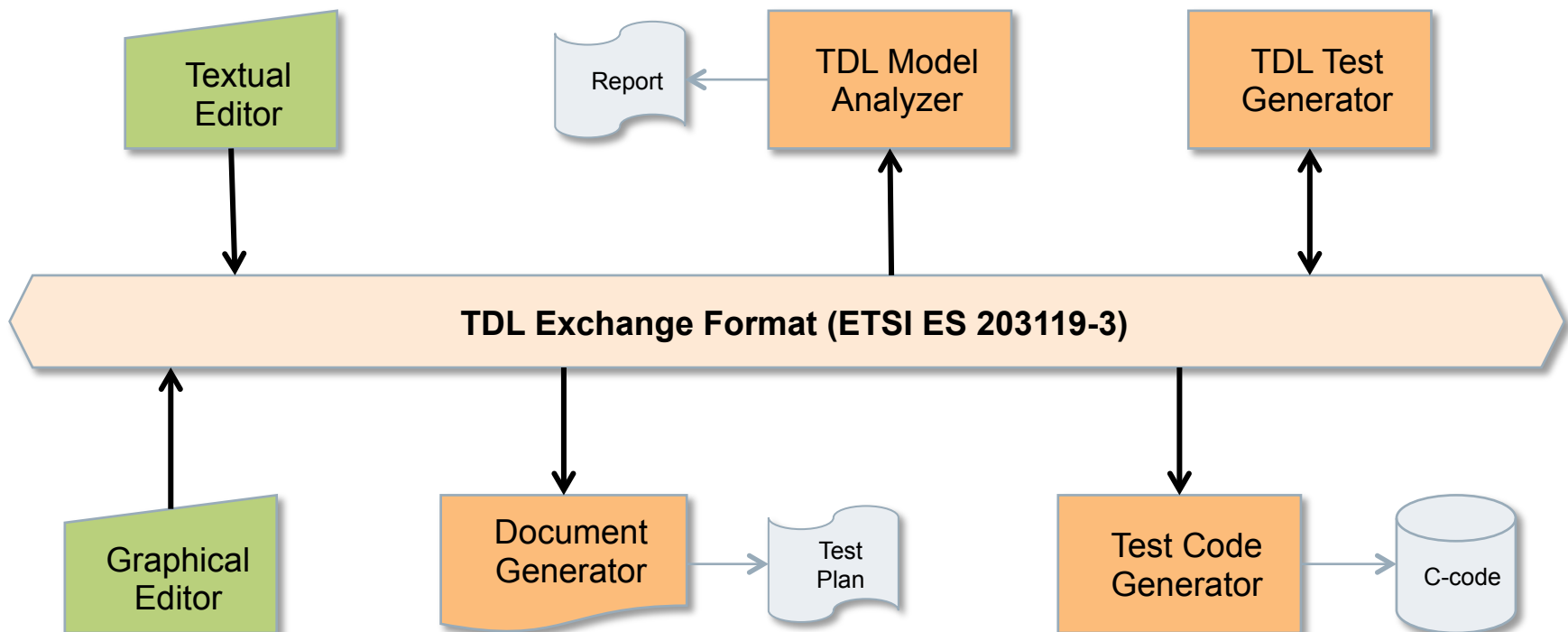
- Design, implementation of textual domain-specific (test) languages
  - Well supported by existing technologies (EMFText, Xtext)
  - Within Eclipse
- Graphical test languages
  - Highly desirable by testers and non-testers for easy visualization
  - Much harder to get tool support

## Recommendations on improving user acceptance for TDL

- Enable a quick start – Provide a TDL reference syntax and implementation
- Tap the potentials of UML editors → UML profile for TDL?
  - Support for graphical notations
  - Easier integration into the development lifecycle process
- Creation of a TDL user group?
  - Exchange forum
  - Social platform

# Outlook: A scalable TDL-based tool architecture

Exchangeable and reusable tool components; adjusted to specific demands





# Contact



**Joachim Fröhlich**  
**Sylvia Jell**  
**Andreas Ulrich**

Siemens AG  
Corporate Technology

Otto-Hahn-Ring 6  
81739 München  
Germany

E-mail:  
[andreas.ulrich@siemens.com](mailto:andreas.ulrich@siemens.com)

Internet:  
[www.siemens.com/innovation](http://www.siemens.com/innovation)