Architecture-driven Test Development[1]

Klaus-D. Engel, Axel Rennoch, Ina Schieferdecker

Fraunhofer FOKUS, MOTION
Kaiserin-Augusta-Allee 31,
10589 Berlin, Germany
www.fokus.fraunhofer.de/en/motion

Abstract:
Industry demands new approaches for test development to overcome the problem of increasing system complexity and efforts for testing. This contribution reflects practical testing conditions in different domains, the development steps for test derivation and emphasises the potentials for testing retrieved from an architectural system definition that has been enriched by pattern and contracts.

Keywords:
System architecture, testing, pattern, contracts

1 Introduction

Describing the architecture of a system deals with describing the components of the system, their interconnection and their possibilities of communicating with each other e.g. by exchanging "information". In the area of mechatronic systems (i.e., systems with mechanical, electronic, and software aspects) this "information exchange" in a system is not only transferring data from one component to another but also transferring energy or material. One may also take into account more details on component offers and requests, i.e. how to describe and find connectable system components and how to connect them to provide composed functionality. This is for example done in the area of Service Oriented Architecture (SOA). Another architectural description style of a system

[1] This work has been partially supported by the ITEA2 project D-MINT (www.d-mint.org).

may focus on communication styles like client-server, publish-subscribe, pipe-and-filter etc.

Modeling systems is in particular getting more complex if they are not limited to one domain like software and have to handle multiple aspects from different domains (e.g. electricity, electronics, mechanics, fluid-mechanics, topological aspects). This forces the developer to separate more and more a lot of aspects into separated views being concerned with specific aspects of the system to be able to handle the complexity of the overall system by dealing with less complex partial aspects. The overall complexity of the whole system has not been reduced but it has been divided into less complex sub-aspects and the problem of there integration.

From the testing point of view, the growing complexity of targeted system under tests (SUTs) demands new methodological ideas for test development to overcome increasing costs or failure probabilities. Section 2 provides information on today's architecture-driven system development. In Section 3 we present an outline on the constraints for test development in different industrial domains and give an overview for test development from our practical work that leads to the idea of architecture-driven test development that is presented with a sample client-server-based system architecture.

2 Architecture-driven development

Within the D-MINT project [M1], we deal with systems from domains like automotive, telecommunication, product engineering and city lights control. The objective to use a system architecture description as a starting point for testing is to get support for the derivation of tests during system development.

In order to cover all these different domains by a common approach for the system architecture description, we agreed that the system architecture should at least consist of the system components [S98], their interfaces (also called ports and being differentiated between required and provided interfaces), the configuration of the system components in the system and their connectors. The interfaces need to be able to represent the following interface styles:

- *Client-server interfaces* defining a set of operations that can be invoked synchronously.

- *Sender-receiver interfaces* allowing the sender to send information to one or more receiver asynchronously, i.e., without awaiting immediate responses from the receivers. A receiver autonomously decides when and how to use the information and if or if not to send a response.
- *Continuous signal interfaces* which work similar to the send-receiver interfaces but transfer "information" as a continuous flow, i.e., at every point in time information is available at the interface.

The "information" being transferred is not necessarily limited to the exchange of data but can also be energy or material.

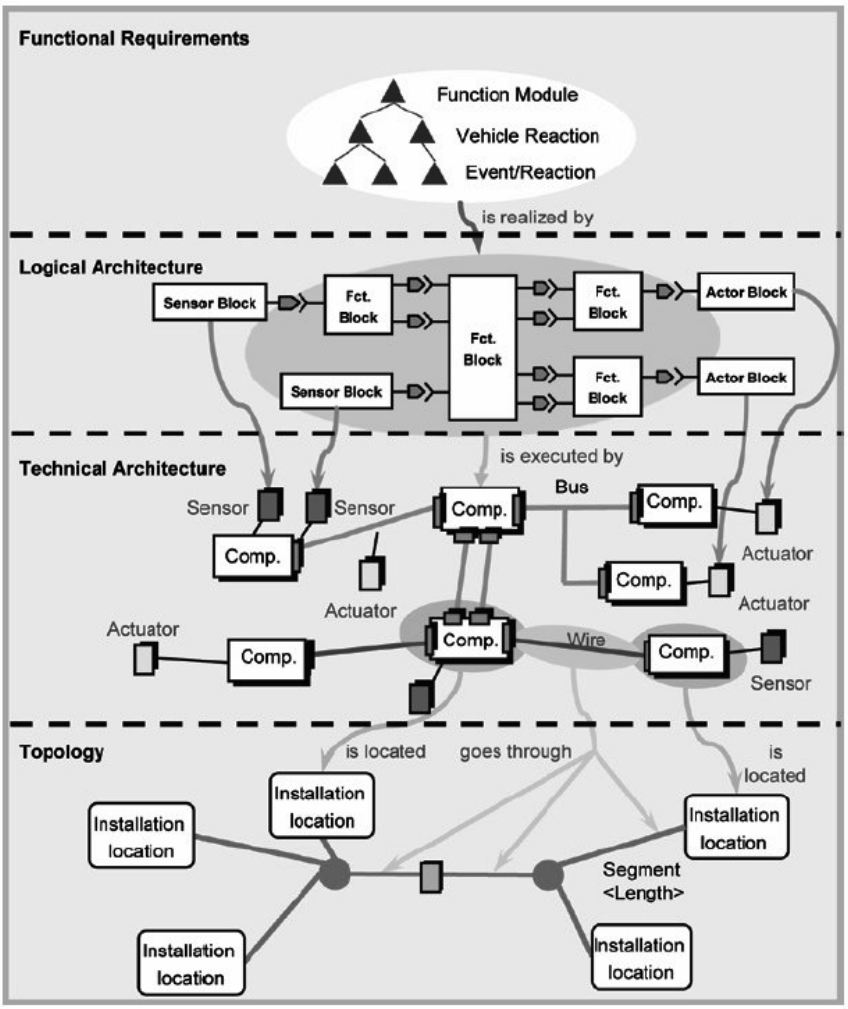

Fig. 1: Modelling layers in the automotive domain [R08]

The system development process creating, using and refining these architectural descriptions will take place in a layered fashion taking into account more and more specific aspects. For the automotive E/E architecture this may look as shown in Figure 1. Starting with the *functional*

requirements, that describe the functions from the view of the customer, the *logical system architecture* describes the network of system blocks, which realize the functions and their interconnection without taking into account any technical realization aspects. These are introduced on the next layers, i.e. the *technical architecture* (hardware elements) and *topology* (space limitations).

For software architectures, a lot of work has been done. A good starting point is the work at [SEI]. There are various architecture description languages and even tools available [S95, F00, GMW00,SOK04]. A lot of them are from the research area. Due to the fact that the approach should be applied in the industrial area we had to focus if possible to industry quality tools for system architecture modelling. Even there we had the selection between generic languages and tools being applicable in multiple areas but not having the specific semantics, expressiveness and specific support like languages and tools being developed for a specific domain or even for a more specific methodology in a company.

One usable language being supported by a lot of even commercial tools is SysML [SML]. It enables a generic approach by modelling the system architecture by a UML profile and supports all kinds of needed interface types (see above). Alternatively, more domain specific languages and tools could be used e.g. for the automotive domain (such as [R08]). They are typically directly accepted in the specific industrial application area.

3 Architecture-driven testing

Industrial domains

Automated test development in industrial domains is based on formal definition on the system or test models. Due to technical and historical reasons, such models strongly depend on the target system characteristics and requirements. In D-MINT work concentrates on case studies in three industrial domains and focuses on a set of particular constraints [M1]:

In Automotive, a current middle-class in-car system demands the design and integration of 60-80 ECUs from different suppliers, with different levels of real-time requirements and many distributed functions. Modelling automotive systems is mainly done with a data-flow based modelling system (e.g. [SM]) with elements and logic for controller design. Test models must include test parameters, system parameters and calibration data in order to be flexible enough to run one and the same test case in all possible configurations of the in-car system. This requires the concept of “parameters” to be incorporated into the test models.

In telecommunication, system models are most often partial specifications only due to the size of the protocol specifications and due to the fact that different standardization organization often have different specification approaches and targets. Furthermore, telecommunication standards often are frameworks that contain several optional requirements and do not intend to specify all implementation details. Hence, the target test model needs to present a variety of different test aspects, like test configuration, test data formats to be exchanged with the SUT, large set of test parameters e.g. due to protocol options or network configuration.

In the industrial automation domain, executable test cases are generated from the test model, i.e., the usage model of a system. A test runner has to be selected to enrich the test model with test automation data, e.g. [IX]. Test cases are paths through the test model, in this case from the starting to the stopping phase with or without restarts. The selection of critical usages and boundary conditions will play a major role for test generation. Failures detected during test evaluation are mapped to test steps and the corresponding transitions and stimuli. A key point is the identification of the correct system boundary, input stimuli and required output responses, to construct a representative usage model, from which one can automatically generate test cases.

From requiremenats to test cases

Reflecting the practical situation of test development in different industrial domains, there is a need for an advanced approach that considers in particular complex system architectures (under test) and an automated tool-support for test generation. The latter issue demands sophisticated algorithms that should be based on practical experiences that we do summarize in Figure 3.

Typically, the test development is a process that consists of a stepwise analysis of the SUT requirements and, in parallel, the creation and refinement of a test model (W-Model approach [S02]). According to the principles of test derivation procedure, test suite structure (TSS) and test purposes (TP) have to be identified first. Elementary test groups (e.g. valid/invalid behaviour) result from the identification of requirements and test methods and types (performance, conformance etc.). The TSS and TP catalogue needs to be enriched due to the identification of entities that are required to interact (trigger or manipulate) with the SUT and may become test components. Such an analysis considers the availability of SUT interfaces and related roles.

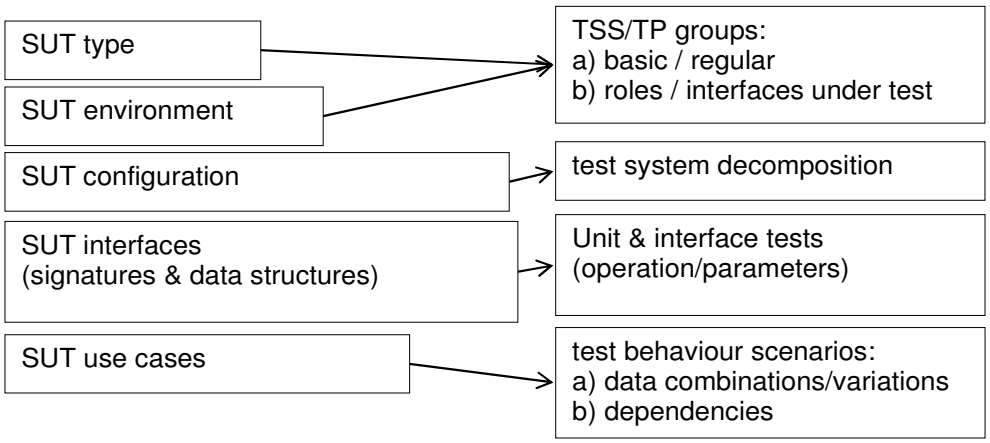

Fig. 3: Test development steps

Corresponding to different SUT configurations under test, the test system (components) may be configured and the TSS&TP refined accordingly. Furthermore, SUT interface operations, i.e. the relevant signatures and parameter structures need to be analysed in order to identify tests per interfaces and primitives and parameter value set. So far, the interfaces (and states/scenarios at single points of control and observation (PCOs)) have been considered individually, but dependencies still need to be investigated. Constraints that exist between multiple SUT interfaces can be described as contracts between software elements [A03] and will lead to further test purpose definitions, including distributed scenarios using correct and invalid test data variations.

The test development process as described above focuses on the underlying system architecture. From theory and practice we know that test behaviour (for example subscription, session establishment or release in a service-oriented system design) follows some typical patterns (such as send-receive or send-discard) that may depend in particular on the application domain. Accordingly, we suppose that the architecture-driven test composition is driven by architectural pattern and contracts between access points (i.e., between communicating peers).

4 Application sample

Following the concepts and principles described in the previous sections, we provide here a small example on our approach for architecture-driven test development. Assume a client-server situation including some contracts between a broker and some agents to be addressed during test execution. Furthermore we make use of (1) request/reply and (2) request/discard pattern to define some initial contractual relations for interacting peers as illustrated in Figure 4.

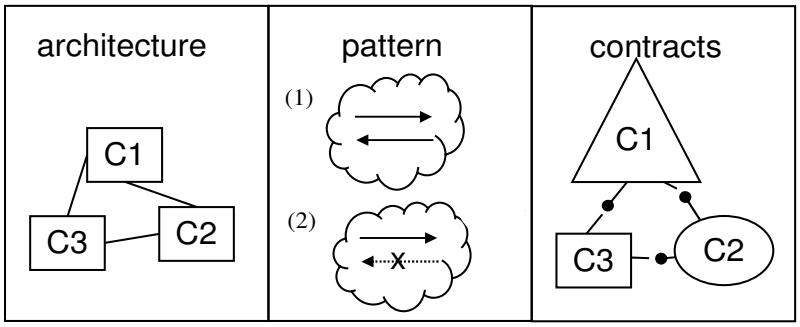

Fig. 4: Sample SUT architectural elements

From the testing point of view, all three interacting components (C1, C2, C3) may be considered as part of the SUT or become part of the test system (and are then to be represented by test components).

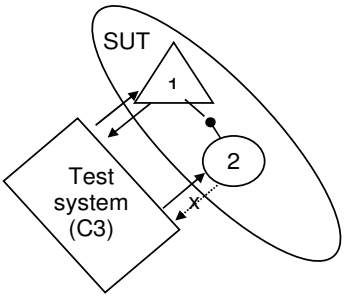

Fig. 5: Sample test configuration

In our example, we select C1 and C2 as the SUT and C3 to be emulated by the test system. Pattern (1) has been selected for the interaction between C1 and C3 and pattern (2) is assumed for the C2/C3 interface. The resulting configuration is provided in Figure 5.

According to our architecture-driven approach and typical test behaviour that reflect our communication pattern, the presented sample configuration (architecture-pattern-contracts) leads us to the simplified test scenarios[2] as described below:

```
P1!request1;
P1?reply1 -> v2;
P2!request2 to v2;
alt{
[] P2?reply2 from v2 {setverdict(fail)}
[else] setverdict(pass)
}
```

[2] A TTCN-3 like abstract test notaion has been used; for simplification we do not give the test description that considers invalid behaviour of the SUT that would require the use of a timer.