



**mehr als nur software**

## **KNOW-HOW 27**

Subsystemtests für Türinterface  
in neuen SMART Aufzügen



**nosser engineering ag**

ch-8404 winterthur  
ch-3048 worblaufen  
ch-6036 dierikon  
d-81369 münchen

talackerstrasse 99  
worblaufenstrasse 163  
industriestrasse 9  
engelhardstrasse 10

tel. 052 234 56 11  
tel. 031 917 52 11  
tel. 041 450 08 11  
tel. 089 7670 4311

fax 052 234 56 22  
fax 031 917 52 22  
fax 041 450 08 22  
fax 089 7670 4312

[www.nosser.com](http://www.nosser.com)

e-mail winterthur@nosser.com  
e-mail bern@nosser.com  
e-mail luzern@nosser.com  
e-mail muenchen@nosser.com

# Subsystemtests für Türinterface in neuen SMART Aufzügen

Unser Kunde die Schindler Aufzüge AG, welcher weltweit als namhafter Produzent von modernen und qualitativ hoch stehenden Aufzugssystemen bekannt ist, verfolgte für die Sicherheit sowie die Zuverlässigkeit der System ein systematische Endprüfung ihrer Produkte.

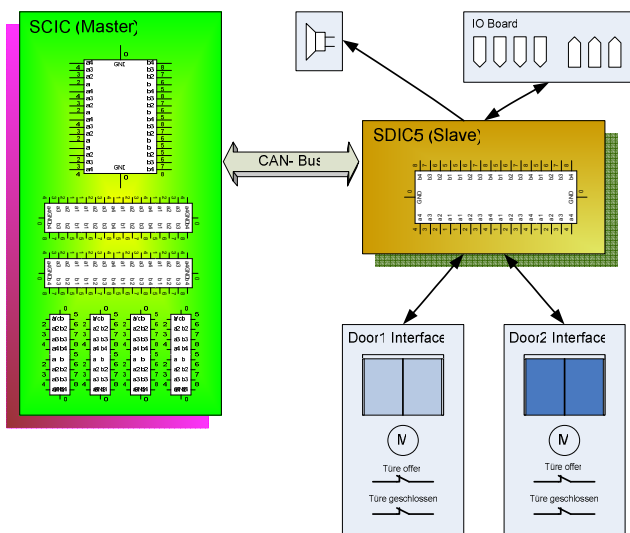
Dieses KNOW-HOW berichtet über die Subsystemtests, des neu entwickelte Türinterface (SDIC5) das in einem künftigen Aufzugssystem eingesetzt wird. Diese Tests wurden als Gesamtpaket unseren Test-Spezialisten anvertraut.

## Das Umfeld

Die Türantriebssteuerung dient als dezentrale Einheit zur Kontrolle, Steuerung und Überwachung der beiden Lifttüren. Diese Steuerung umfasst:

- Das SDIC5 board (HW & SW neu)
- Diverse Anschlüsse für die Türe und
- Viele Optionen

Die Steuerung (SDIC5) kann zwar als externe Einheit gewisse Aufgaben komplett losgelöst vom Master übernehmen und ausführen, im eigentlichen System fungiert sie jedoch als reiner Slave.



Vereinfachte Darstellung vom Gesamtsystem

## Der Auftrag

Erfahrungsgemäss stellte das Türinterface immer einen der heikelsten Punkte im Gesamtsystem dar, was meist zu einem mehrmaligen Systemtestdurchlauf führte, der sich im Umfang von bis zu drei Monaten bewegte. Auf diesem Hintergrund hat man sich entschieden, das Türinterface mit gezielten Subsystemtests vorzutesten.

## Unsere Aufgabe

Unsere konkrete Aufgabe war es gezielte und fundierte Subsystem Testcases zu designen und diese zu automatisieren.

## Die Test-Strategie

Die Teststrategie wurde zusammen mit unserem Kunden und den Entwicklern erarbeitet und umfasste nachfolgendes Vorgehen:

1. Parallel zum Usecase Design der Entwicklung, wird bereits mit dem Design der dazupassenden Testcases begonnen. Somit werden die Usecases automatisch von einer unabhängigen Testinstanz mit reviewed.
2. Die Testcases werden so entwickelt, dass alle vorhandenen Funktionen komplett abgedeckt werden; dies entspricht einer systematischen Testabdeckung.
3. Die Testcases werden wie die Usecases priorisiert (high, med und low). Alle high und med, Testcases werden automatisiert und auf Subsystem Level mit jedem Build getestet.
4. Für das Erreichen einer automatischen Rückwärtskompatibilität müssen die Testcases auf der neuen sowie alten Testumgebung ausgeführt werden können.
5. Das Türinterface wird erst im Gesamtsystem integriert, wenn alle Testscripts fehlerfrei den Subsystemtest bestanden haben. Mit diesem Vorgehen sind diese Subsystem Tests einen Bestandteil des Integrations-Prozesses.

## Die Realisierung der Testcases

Das parallele Entwickeln der Testcases zu den Usecases erwies sich als sehr ergiebig, da das „Systemtesting Argusauge“ steht's auf die Usecases der Entwickler fokussiert war. Umgekehrt konnte wir unheimlich viel von den Entwickler über die „Innereien“ der Software und dessen Design in Erfahrung bringen.

Die enge Zusammenarbeit schuf auch einen nicht zu unterschätzenden wichtigen sozialen Zusammenhalt, der gerade bei verteilten Entwicklungen<sup>1</sup> einen unschätzbaren Wert darstellen kann.

Somit waren die Testcases praktisch zeitgleich zu den Usecases fertiggestellt und reviewed gewesen. Dies repräsentierte schon ein erstmaligen wesentlichen Zeitgewinn im gesamten Entwicklungsprozess.

<sup>1</sup> Die Entwicklung der Usecase erfolgte in Ebikon und Locarno

## Die Realisierung der Testumgebung

Da wir das SDIC Interface als Subsystem losgelöst von allen Systemkomponenten betrachten wollten, mussten die fehlenden Komponenten in unserer Testumgebung nachgebildet werden:

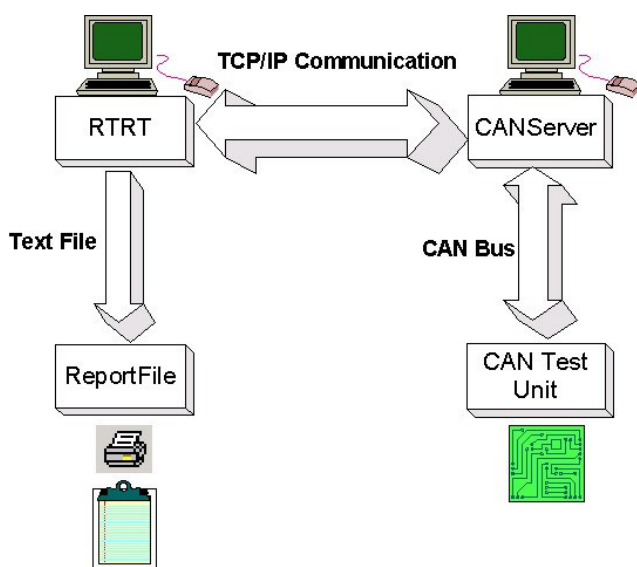
1. Master
2. IO's und
3. Türsimulator

Es zeigte sich jedoch, dass keine annähernd befriedigende käufliche Lösung zu finden war, was dazu führte eigene Testumgebung mit allen benötigten IO's und den beiden Türsimulatoren aufzubauen.

Für die Automatisierung, welche nur auf den Messages des CAN<sup>2</sup>-Bus erfolgte, wurde das Testtool RTRT<sup>3</sup> verwendet. Dazu wurden die nachfolgenden spezifischen Adaptionen für das Tool sowie die Testumgebung implementiert:

1. CAN Server, welcher die Befehle vom Master in CAN- Messages und die Rückmeldungen vom Slave in TCP / IP Protokolle umwandelt
2. Eigenes Reporting für die Testscript's
3. IO's und deren Zustände mussten ebenfalls in einem Testscript verwendbar sein

Die kommende Abbildung zeigt die Funktionsweise der von uns erstellten Testumgebung:



Übersicht der Testumgebung basierend auf RTRT

## Rational Test RealTime

Das Tool RATIONAL Test RealTime ist ein Bestandteil des RATIONAL RealTime Produktpalette. Es kann aber natürlich auch als einzelne Applikation eingesetzt werden.

<sup>2</sup> CAN = Controller Area Network; © by Robert Bosch GmbH

<sup>3</sup> RTRT = Rational Test Real Time; © by IBM

RTRT kann für Unit / Subsystemtest, Integrationstest und Systemtests eingesetzt werden.

Dank seinem *Message Adaptation Layer* Konzept kann es jede Art von Schnittstellen unterstützen und mit Hilfe der *Custom TestScript* können sich eigene Testprozeduren in das Testscenario eingebunden und mehrfach werden.

Da wir in unserem Projekt RTRT für die Subsystemtest einsetzen, erledigte das Tool die folgenden Aufgaben:

- Master Funktion,
- Ausführung der Testscripts und
- Erzeugen des Reportfiles

## Die Realisierung der Automatisierung

Mit dem plattformunabhängigen Design der Testscripts erlangte man die Möglichkeit, die implementierten Testcases bereits im Vorfeld auf älteren Testumgebung (SDIC3) auszutesten und war somit unabhängig von der zu diesem Zeitpunkt noch nicht zur Verfügung stehender neuen Applikation. Diese Unabhängigkeit verschaffte einen zusätzlichen Zeitgewinn.

Neben diesem Zeitgewinn erlangte man weitere zwei Vorteile:

1. Die Rückwärtskompatibilität zur alten Hardware und Software wurde sichergestellt und
2. die getesteten Testscripts konnten gleich für die Integration der neue implementierten Funktionen verwendet werden

## Testscript Entwicklung

Die Automatisierung basierte ausschliesslich auf den CAN-Messages, welche sehr übersichtliche und gut verständlich implementiert werden konnten:

```
tc4.pts - Notepad
File Edit Format Help

SCENARIO TC4
  | -- Title: Open Door_1 at normal speed
SCENARIO PRECONDITION
  CALL starttestcase("TC4")
  CALL phs_phuet(1) --
  CALL phs_rising() --
END SCENARIO
SCENARIO TRIGGERS
  CALL open_command_door1() --
END SCENARIO
```

Codeauszug von einem Testscript

Der modulare Aufbau der Testscripts ermöglicht die folgenden Anforderungen:

1. Klar und definierte Anfangsbedingungen
2. Der Trigger
3. Der eigentliche Test
4. Abschluss des Tests und Abschlussbedingungen

## Interner Rückblick

Dank einer stabilen Testumgebung und der sehr einfachen Skriptsprache können Testscripts von Entwickler in Zukunft selber geschrieben werden. Wir stellten mit Freuden fest, dass dies bereits geschehen ist und die Entwickler ihre Software auf dieser Testumgebung integrieren und vortesten!

Das Projekt: „Automated Subsystemtests: SDIC5 with RTRT“ wurde im Juni 2004 abgeschlossen und unterstützte aktiv die termingerechte Freigabe.

Das Gesamtsystem BIONIC5 erreichte die Verkaufsfreigabe im geplanten und vorgegebenen Zeitrahmen. Dies war unter anderem nur möglich, dank unserer Teststrategie, die sich in allen Punkten bewährt hatte!

1. Es resultierte keinen zeitlichen Verzug nach der Usecase Design Phase, da die Testcases praktisch zum selben Zeitpunkt fertig gestellt wurden
2. Die Testcases waren erfolgreich und konnten die Schwachstellen der Software aufdecken
3. Der Aufwand für die Automatisierung hat sich mehrfach bezahlt gemacht
4. Mit den Testscripts konnten:
  - Kompatibilitätstest,
  - Integration und schliesslich
  - Subsystemtestgemacht werden
5. Der sehr enge Kontakt mit den Entwickler trug zu einem sehr fruchtbaren und ergiebigen „win- win“ Situation bei.

Noser Engineering AG ist ein unabhängiges Dienstleistungsunternehmen im Gebiet der technischen Engineering-Leistung, welche von der individuellen Software-Entwicklung bis hin zu komplexen System-Tests reicht.



Autor: Florian Rösli  
Dipl. El. Ing. HTL

## Aus Sicht des Kunden

*Seit Frühjahr 2000 arbeitet Micamation AG mit der Firma Noser Engineering AG zusammen, zuerst mit Peter Rutschmann und jetzt mit Mark Lusti als Projektleiter.*

*Noser unterstützt unsere Softwareabteilung und widmet sich vor allem der Verbesserung der bestehenden Software sowie Weiterentwicklungen und Lösungen für neue Maschinen.*

*An regelmässigen Sitzungen der Firma Noser wird der Stand der Arbeiten festgehalten und neue Aufgaben für die Softwareentwicklung definiert. Vorbildlich wird uns jeden Monat ein Bericht der laufenden Arbeiten, Pendenzen, weiteres Vorgehen und Terminsituation zugestellt.*

*Fazit unserer Zusammenarbeit mit der Firma Noser:*

*Ein kompetenter und zuverlässiger Partner mit dessen Unterstützung wir uns auf dem spezialisierten Markt weltweit an der Spitze halten können.*

*Guido Kühne  
Mitinhaber und Geschäftsleiter  
der Micamation AG*