Modellbasiertes Testen zeitbezogener LST-Anforderungen unter Berücksichtigung der RCA

Die automatische Generierung von Testfällen aus Verhaltensmodellen von Systemen kann die Effizienz und Qualität des Designs von Tests standardmäßiger funktionaler Anforderungen steigern. Anhand eines konkreten Modells und Testfallgenerators wird der Frage nachgegangen, wie sich daneben auch Tests typischer zeitbezogener Anforderungen an Leit- und Sicherungstechnik generieren lassen. Zugleich werden Erfahrungen mit der Modellierung gemäß „Reference CCS Architecture“ gesammelt.

1. Einleitung

1.1. Modellbasiertes Testen

Modellbasiertes Testen (MBT) bezeichnet die Nutzung von Modellen als Teil von Testprozessen mit dem Ziel, die Generierung von Testartefakten zu automatisieren. Meist sind es Testfälle für funktionale Anforderungen, die generiert werden [1]. Ein geeignetes Modell für einen Testfallgenerator zu erstellen kann herausfordernd sein; es als zentralen Bezugspunkt der Testfallerstellung zu verwenden, bringt jedoch unter anderem Vorteile bei

- der Qualität (z. B. Konsistenz) und Wartbarkeit der Testsuite,
- der Nachvollziehbarkeit der Testauswahl (definierte strukturelle Modellabdeckung durch die Testsuite),
- der Wiederverwendbarkeit (z. B. für Systemvarianten) sowie
- der Kommunikation (z. B. zwischen Testern, Entwicklern und Gutachtern).

Diese Vorteile machen MBT auch für Systeme der Leit- und Sicherungstechnik (LST) attraktiv, die bekanntermaßen einer ausführlichen und nachvollziehbaren Validierung und Verifikation unterliegen [2, 3, 4], um dauerhaft für sicherheitskritische Aufgaben einsetzbar zu sein. In [5] wurde die Anwendbarkeit am Beispiel eines einfachen reaktiven Systems – des EULYNX Subsystems Point [6] – demonstriert. „Einfaches reaktives System“ bedeutet hierbei, dass der durch EULYNX spezifizierte Weichencontroller als System verstanden wird, das auf eingehende Befehle vom Stellwerk oder Meldungen der Weichenantriebe wartet, um dann seinen internen Zustand zu wechseln und ausgehende Befehle an den Weichenantrieb oder Meldungen an das Stellwerk zu senden. Dabei wird ein eingehendes Ereignis vollständig abgearbeitet, bevor ggf. das nächste eingegangene Ereignis behandelt wird.

Für solche Systeme lassen sich Testfälle generieren, die die meisten funktionalen Systemanforderungen überprüfen. Typischerweise haben diese Anforderungen die Struktur

WENN <eingehendes Ereignis> IN <aktuellem Zustand> DANN <neuer Zustand> UND <ausgehende Ereignisse>.

Ein passender Testfall zu dieser Anforderung stellt also zunächst ggf. in mehreren Schritten den gewünschten aktuellen Zustand her, sendet dann das eingehende Ereignis an das zu testende System (engl. system under test – SUT) und prüft, ob das System die richtigen ausgehenden Ereignisse sendet. Die Systemzustände werden dabei implizit mitgeprüft.

1.2. Zeitbezogene Anforderungen

Bereits im Beispiel des Weichencontrollers zeigt sich jedoch, dass nicht alle funktionalen Anforderungen auf diese Weise testbar sind: der Controller besitzt – wie viele andere LST-Systeme auch –

1. (SUT-)Timer, deren Ablauf nach einer festgelegten Zeit (analog zu einem eingehenden Ereignis) interne Zustandswechsel und ausgehende Ereignisse auslösen kann.

Entsprechende Systemanforderungen haben die Form

WENN <ablaufender Timer> IN <aktuellem Zustand> DANN <neuer Zustand> UND

Dr. Daniel Schwencke[1]
Wissenschaftlicher Mitarbeiter am Deutschen Zentrum für Luft- und Raumfahrt e.V. (DLR), Institut für Verkehrssystemtechnik; Forschung zu Verifikations- und Validierungsmethoden für Leit- und Sicherungstechnik
daniel.schwencke@dlr.de

[1] Die Arbeiten wurden im Rahmen des Projekts X2Rail-5 durchgeführt. Das Projekt wurde durch das Shift2Rail Joint Undertaking innerhalb des Horizon 2020 Forschungs- und Innovationsprogramms der europäischen Union unter der Zuwendungsvereinbarung Nr. 101014520 gefördert.

<ausgehende Ereignisse> UND <Timerstarts/-stopps>.

Über die bloße Reihenfolge eingehender Ereignisse ist nun auch der relative Zeitpunkt ihres Eintreffens von Bedeutung, um zu steuern, an welchen Stellen zwischen den eingehenden Ereignissen Timerabläufe im SUT auftreten. MBT-Werkzeuge zur Generierung von Testfällen wie IBM Rhapsody ATG [7], RT-Tester [8] oder Conformiq Designer [9] können hiermit jedoch meist umgehen, indem Verzögerungselemente in die Testfälle hineingeneriert werden, die bei Testausführung dafür sorgen, dass der Timerablauf im SUT an der gewünschten Stelle ausgelöst wird.

Darüber hinaus spielt Zeit beim Testen (von LST-Systemen) für weitere Aspekte neben SUT-Timern eine Rolle, bei denen nicht unmittelbar klar ist, wie sie in der MBT-Praxis behandelt werden können:

2. Umgebungstimer. Oft ist es sinnvoll, neben dem SUT auch die Testumgebung zu modellieren, so dass der Testfallgenerator nicht komplett beliebige eingehende Ereignisse erzeugt, sondern auf weitgehend realistische Szenarien beschränkt wird. Dabei können analog zur Modellierung des SUT Timer verwendet werden.
3. Zeitliche Begrenzung von Negativtests. Gelegentlich gibt es Anforderungen, die spezifizieren, dass etwas (das Senden eines ausgehenden Ereignisses) NICHT geschehen soll. Um diese sinnvoll testen zu können (Negativtests), braucht es eine Begrenzung, bis wann die Anforderung gilt. Diese Begrenzung kann durch ein Zeitintervall gegeben sein.
4. Maximale Systemlaufzeiten. Neben funktionalen gibt es auch nicht-funktionale Anforderungen. Bei LST-Systemen finden sich hier häufig geforderte maximale Verzögerungen durch das SUT vom Eingang eines Ereignisses (oder Ablauf eines Timers) bis zum Senden eines ausgehenden Ereignisses, die getestet werden müssen.
5. Systemverhalten unter Last. Ebenfalls als nicht-funktionale Anforderung ist oft gefordert, dass das System eine gewisse Anzahl von in einer bestimmten Zeit eingehenden Ereignissen korrekt abarbeiten kann (und nicht etwa vorher Ereignisse verwirft, abstürzt o. ä.). Hierfür werden Lasttests vorgesehen.
6. Ermittlung der Systemkapazität. Über die Erfüllung von Anforderungen hinaus kann es interessant sein (Einschätzung der Robustheit, Marketing) zu ermitteln, wie viele eingehende Ereignisse pro Zeit das System abarbeiten kann, ohne nennenswerte Einbußen bei Funktionalität oder Performance zu zeigen. Hierzu werden Kapazitätstests ähnlich den Lasttests verwendet, jedoch als Testserien, innerhalb derer die Anzahl der eingehenden Ereignisse pro Zeit variiert wird.

Folgende Aspekte sollten allgemein bei der Konzeptionierung modellbasierten Testens von zeitbezogenen Anforderungen bedacht werden:

- Für das Testen solcher Anforderungen ist – gerade bei den oft stark konfigurationsabhängigen LST-Systemen – zu beachten, dass das Zeitverhalten eines SUT abhängig von der Konfiguration des SUT sein kann. Daher ist jeweils eine geeignete Systemkonfiguration zu wählen, für die die Tests generiert werden.
- Im Detail hängen mögliche Lösungen vom genutzten Testfallgenerierungswerkzeug ab, insbesondere von den unterstützten Modellkonstrukten.
- In der Regel werden im Rahmen von MBT diskrete Modelle von Zeit verwendet. Kontinuierliche Zeit ist schwieriger zu behandeln und wird nur selten (z. B. in [10]) thematisiert.

1.3. Artikelüberblick und Beispielmodell

Die in Abschnitt 1.2 genannten Arten von zeitbezogenen Anforderungen werden im Folgenden in Abschnitt 2 anhand von Beispielen auf ihre Eignung für modellbasiertes Testen untersucht. Die Beispiele stammen aus einem durch die Reference CCS Architecture (RCA; CCS = command control and signalling) Spezifikationen [11] inspirierten Modell, das mit dem Werkzeug IBM Rhapsody [12] in der Systems Modeling Language (SysML) [13] erstellt wurde und aus dem mit Hilfe des Rhapsody ATG Add-Ons [7] Testfälle generiert wurden. Es wurde eine klassische blockbasierte LST mit Signalen zur Modellierung gewählt; während die optimale Unterstützung neuerer / künftiger Betriebsverfahren und LST sicherlich eine Motivation für die Entwicklung von RCA [14] waren / sind, schien es interessant, wie gut klassische Systeme sich (noch) damit abbilden lassen. In Abschnitt 3 werden die Erfahrungen mit der RCA-geleiteten Modellierung knapp reflektiert.

Als Referenz für die weiteren Ausführungen dieses Artikels bietet Bild 1 einen groben Überblick über die Architektur und den Funktionsumfang des Modells. Es unterstützt unterschiedliche Konfigurationen (Gleispläne, Signalanordnungen), die durch die Komponente „Device and Config Management" an die beiden Hauptkomponenten „Plan Execution" und „Safety Logic" des Systems gesendet werden. Aufgabe von „Plan Execution" ist es vor allem, Züge und die für sie vorgesehenen Fahrwege zu verwalten und die Fahrwege umzusetzen. Hierzu werden Verfügbarkeit und Status benötigter Fahrwegelemente geprüft und ggf. Änderungen von Weichenpositionen und / oder Signalbegriffen bei „Safety Logic" angefragt. Dort besteht die Hauptaufgabe darin zu prüfen, ob die Änderung zulässig ist und bei positivem Prüfergebnis ein Kommando an die jeweiligen Weichen / Signale zu senden. Die Prüfung wird auf Basis des Status von „Drive Protection Sections" (DPS) durchgeführt, die Gleisabschnitte wie z. B. einen Weichenstrang darstellen, die nicht befahrbar, befahrbar oder zum Befahren reserviert sein können. Auch die Aufhebung der Reservierung und das Zurückstellen der Signale auf Halt (i. d. R. nach Passieren des jeweiligen Elements durch den Zug, bei Signalen ggf. auch auf ein Bedienkommando hin) sowie die Verwaltung von Durchrutschwegen erfolgt durch „Safety Logic".

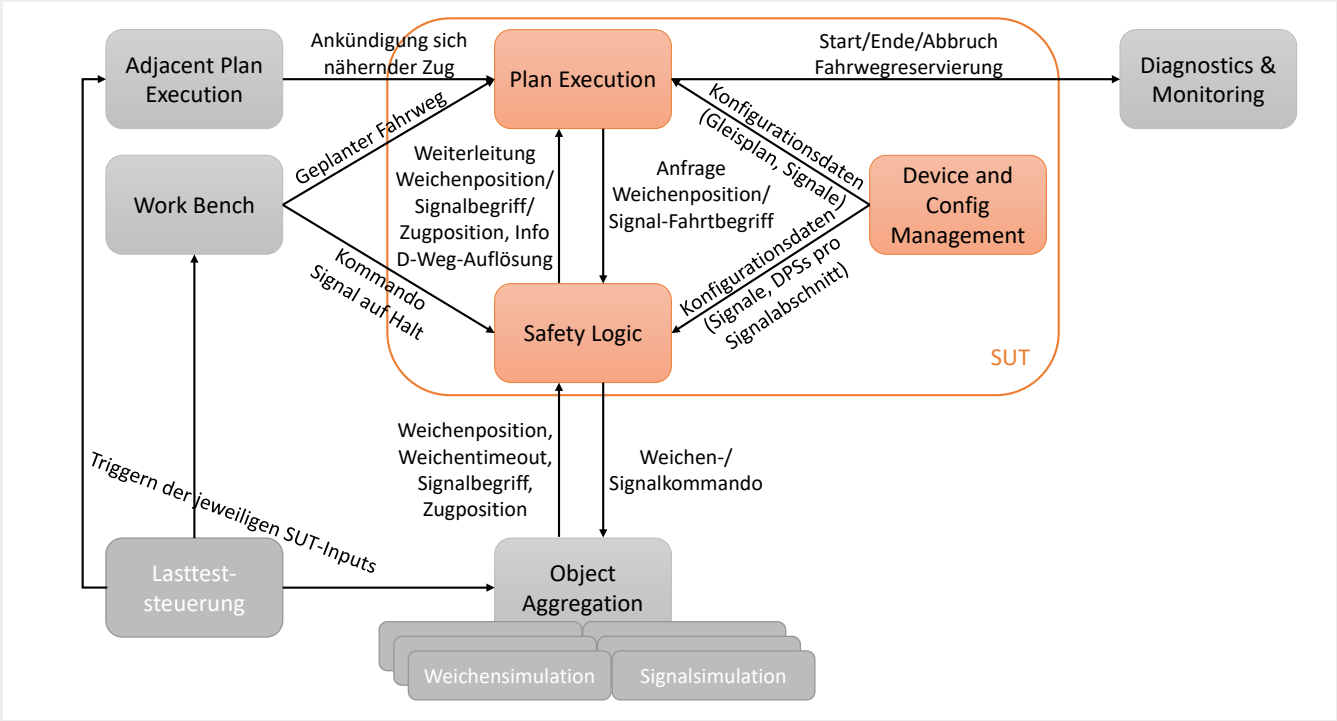
2. Testfallgenerierung für zeitbezogenes Verhalten

Für jede Art zeitbezogener Anforderungen aus Abschnitt 1.2 werden im Folgenden Beispiele aus dem modellierten LST-System unter Angabe der betroffenen Modellkomponente (Bild 1) genannt. Anschließend wird die gewählte praktische Umsetzung der Anforderungsart erläutert und diskutiert.

2.1. SUT-Timer

Beispielanforderungen

1. (Plan Execution) Abbruch einer nicht erfolgreichen Anfrage von Fahrwegelementen (Weichen, Signale) nach einiger Zeit. Der Timer wird gestartet, sobald die Fahrwegelemente reserviert wurden und ihr gewünschter Status (bei Safety Logic) angefragt wurde. Er wird gestoppt, sobald alle Elemente den gewünschten Status (via Safety Logic) zurückgemeldet haben oder die Anfrage zurückgezogen wurde. Wenn er abläuft, wird die Reservierung der Fahrwegelemente zurückgezogen.



1: Architektur des Testfallgenerierungsmodells – in orange die SUT-Komponenten, in grau die Komponenten der Testumgebung. Die englische Benennung der orangen und hellgrauen Komponenten entspricht der Benennung der durch RCA definierten Architekturkomponenten; die deutsch benannten dunkelgrauen Komponenten stellen nicht durch RCA definierte Komponenten der Testumgebung dar Quelle: DLR

2. (Safety Logic) Zeitgesteuerte Durchrutschwegauflösung. Der Timer startet, sobald der Zug seinen letzten Fahrwegabschnitt – d.h. an dessen Ende ein haltzeigendes Signal steht – befährt. Er wird gestoppt, falls der Durchrutschweg anderweitig aufgelöst wird, z.B. bei einer Fahrwegverlängerung. Wenn er abläuft, wird der Durchrutschweg aufgelöst.

Umsetzung

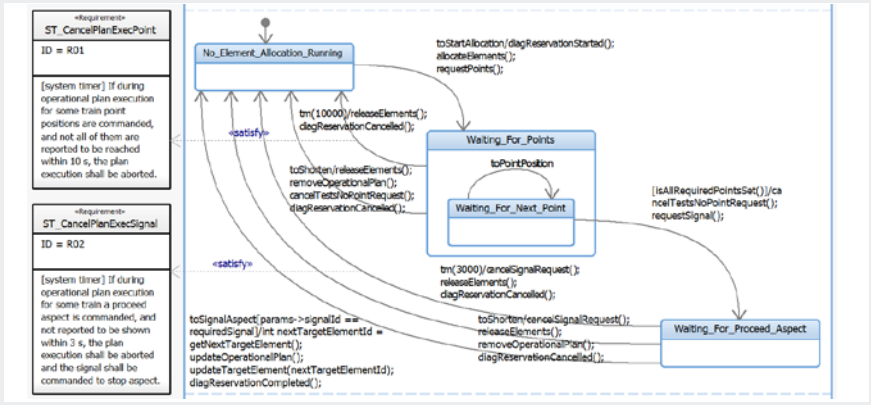
Im Modell werden Time-out-Transitionen zwischen zwei Zuständen genutzt, s. Bild 2. Beim Betreten des Startzustandes wird der Timer gestartet; wenn er abläuft, wird die Time-out-Transition getriggert (jedoch nur ausgeführt, wenn mögliche weitere Transitionsbedingungen erfüllt sind); wenn zuvor der Zustand über eine andere Transition verlassen wird, wird der Timer gestoppt.

Im Testfall wird für den Fall, dass der Timerablauf nach x Millisekunden getestet werden soll, nach den Schritten, die in den Startzustand führen, ein (Mindest-)Zeitintervall von x ms generiert, und danach die Schritte, die Konsequenz des Timerablaufs sind (typischerweise SUT-Outputs), s. Bild 3. Für den Fall, dass ein SUT-Input den Timer vor dessen Ablauf stoppt, werden die Schritte für den Input und die Konsequenzen dieses Inputs ohne Zeitintervall direkt im Anschluss an die Schritte, die in den Startzustand führen, generiert.

Diskussion

Beim Test des Timerablaufs kann durch das generierte Zeitintervall überprüft werden, dass die notwendige Bedingung (>= x ms) für das Ablaufen des Timers sowie die Konsequenzen des Timerablaufs vorliegen. Eine Prüfung, dass der Timerablauf nicht zu stark verspätet eintritt, kann in der Testumgebung leicht ergänzt werden, beispielsweise unter Nutzung eines allgemein erlaubten Zeitintervalls für Timerabläufe.

Beim Test des Timerstopps fehlt die Information, wann der auslösende SUT-Input



2: Umsetzung der ersten SUT-Timer-Beispielanforderung im Modell durch Time-out-Transitionen (die zwei Transitionen, deren Label mit „tm(…)" beginnt und die mit den links dargestellten Anforderungen verknüpft sind): nach 10.000 ms bzw. 3000 ms erfolglosem Warten auf die korrekten Weichenpositionen bzw. den Fahrtbegriff des Signals wird die Fahrweganfrage abgebrochen und in den Ausgangszustand zurückgekehrt Quelle: DLR

durch den Testtreiber gesendet werden muss. Eine präzise Angabe einer Testumgebungsbedingung aus der SUT-Bedingung „< x ms bis zum Empfang des Inputs“ abzuleiten ist ohnehin nicht möglich, da die Laufzeit des Inputs zum SUT unbekannt ist. Bedenkt man außerdem, dass ein Testtreiber in der Regel[2] ohnehin den nächsten Input so bald wie möglich senden würde, um die Testausführungszeit zu minimieren, und ein verfrühtes Ablaufen des SUT-Timers bereits durch eine Verletzung der Bedingung >= x ms aus dem Timerablauf-Testfall detektiert würde, ist diese Information auch nicht nötig.

2.2. Umgebungstimer

Beispielanforderungen

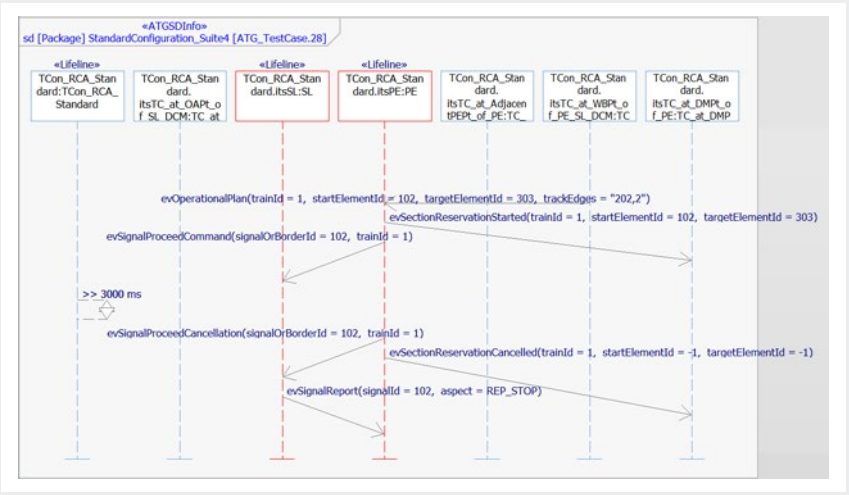
1. (Weichensimulation) Senden einer Time-out-Nachricht (via Object Aggregation an Safety Logic), wenn eine kommandierte Endposition der Weiche nicht innerhalb einer definierten Zeit erreicht wird (Simulation eines EULYNX-konformen Weichencontrollers, vgl. [6]).
2. (Weichensimulation) Senden der erreichten Endposition (via Object Aggregation an Safety Logic) mit einer gewissen Verzögerung nach dem Erhalt eines Weichenkommandos (Simulation der Weichenumlaufdauer).
3. (Signalsimulation) Senden des angezeigten Signalbegriffs (via Object Aggregation an Safety Logic) mit einer gewissen Verzögerung nach dem Erhalt eines Signalkommandos (Simulation der Signalstelldauer).

Umsetzung

Die Umsetzung erfolgt analog zur in Abschnitt 2.1 beschriebenen Umsetzung von SUT-Timern. Die im Testfall befindlichen Konsequenzen nach dem generierten Zeitintervall (s. das Intervall in Bild 4) bzw. nach einem den Timer stoppenden Umgebungsinputs sind nun jedoch typischerweise SUT-Inputs.

Diskussion

Timerablauf und Timerstopp werden nicht explizit getestet, da sie nicht Teil des SUT sind. Sie können jedoch als modelliertes Umgebungsverhalten in den Testfällen vorkommen. Beim Timerablauf kann durch das generierte Zeitintervall das Zeitverhalten der Umgebung während des Tests berücksichtigt werden (Auslösen einer Umgebungsreaktion mit >= x ms Verzögerung). Würde beim Timerstopp die Aussage „< x ms bis zum Empfang des stoppenden Umgebungsinputs“ in den Testfall generiert, ließe sich diese zwar während des Tests durch die Testumgebung prüfen, es ließe sich jedoch keine präzise Aussage über das SUT daraus ableiten, selbst wenn das SUT der Sender des Umgebungsinput wäre. Von daher ist es auch hier sinnvoll, dass diese Information nicht generiert wird.



3: Ausschnitt aus einem generierten Testfall als SysML-Sequenzdiagramm mit 3000 ms Zeitintervall entsprechend der zweiten Time-out-Transition aus Bild 2. Unmittelbar vor dem Intervall wird ein Fahrtbegriff durch Plan Execution (rechte rote Linie) bei Safety Logic (linke rote Linie) angefragt. Da jedoch keine Antwort erfolgt, erwartet der Testfall nach Ablauf der im Intervall angegebenen Zeitspanne, dass die Anfrage abgebrochen wird Quelle: DLR

2.3. Zeitliche Begrenzung von Negativtests

Beispielanforderung

(Plan Execution) Keine erneute Anfrage einer Weichenposition an Safety Logic für die gleiche Weiche innerhalb einer Zeitspanne nach einer Anfrage (es sei denn, der zugehörige Fahrweg wurde inzwischen fertig eingestellt oder zurückgezogen).

Umsetzung

Für das verbotene Verhalten wird im Modell ein entsprechender SUT-Output definiert, der mit dem Schlüsselwort „_NOT_DURING_“ beginnt, mindestens einen ersten Parameter hat, der die Zeitspanne des Verbots angibt, und ggf. weitere Parameter besitzt um den verbotenen Output weiter zu spezifizieren. Das Senden des Outputs wird im betrachteten Verarbeitungspfad des SUT nach dem In- oder Output modelliert, der den Startzeitpunkt des Verbots definiert. Der Testfallgenerator generiert diesen speziellen Output wie jeden anderen Output in die Testfälle (s. Bild 4 im oberen Bereich). Für Pfade, auf denen der Negativtest abgebrochen werden soll, wird der gleiche Output noch einmal mit der Zeitspanne 0 gesendet (s. Bild 4 im unteren Bereich).

Diskussion

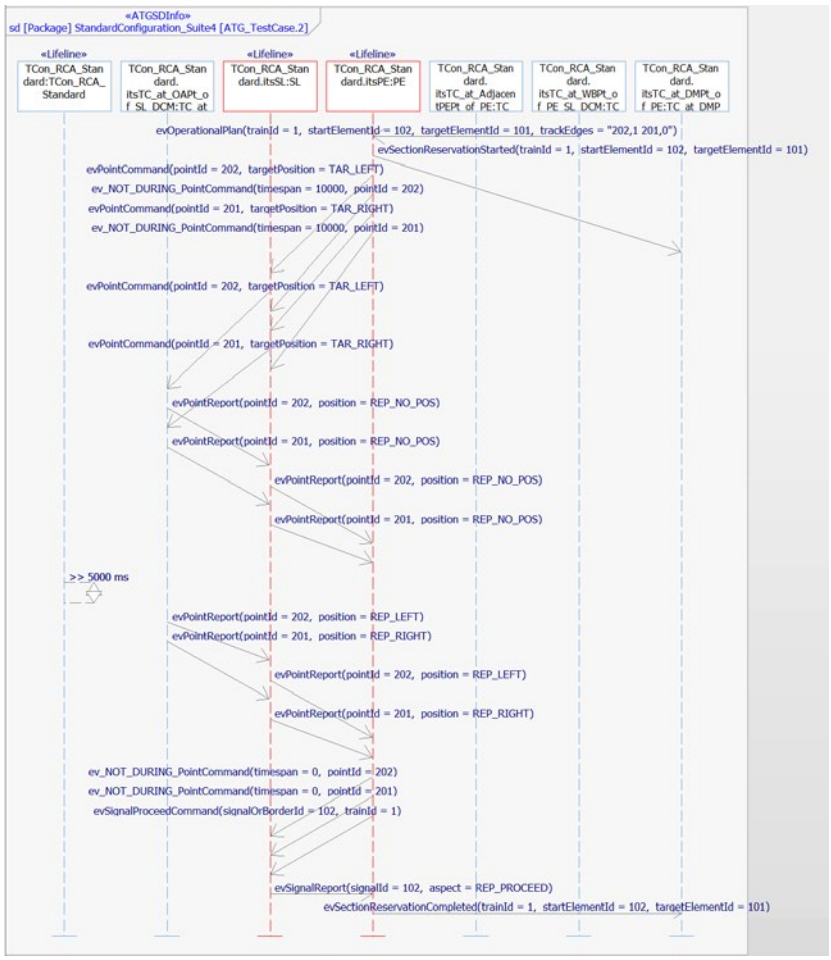
In Modellen für die Testfallgenerierung wird in der Regel das erlaubte Verhalten modelliert und entsprechend werden nur Positivtests durch die Testfallgeneratoren erzeugt. Die obige Umsetzung erfindet daher pragmatisch eine Klasse von zeitlich begrenzt verbotenen Outputs, die während der Abarbeitung der Tests anhand des Schlüsselworts als solche erkannt und überprüft werden. Diese Outputs bereits ins Modell einzubauen hat den Vorteil, dass der Negativtest systematisch in die Testfälle mitgeneriert wird; es hat den (kleinen) Nachteil, dass dies fälschlicherweise suggeriert, dass beim Test geprüft wird, dass das SUT keine entsprechende Nachricht innerhalb der angegebenen Zeitspanne sendet, während (leicht abweichend davon) real geprüft wird, dass keine solche Nachricht innerhalb der Zeitspanne durch die Testumgebung empfangen wird.

2.4. Maximale Systemlaufzeiten

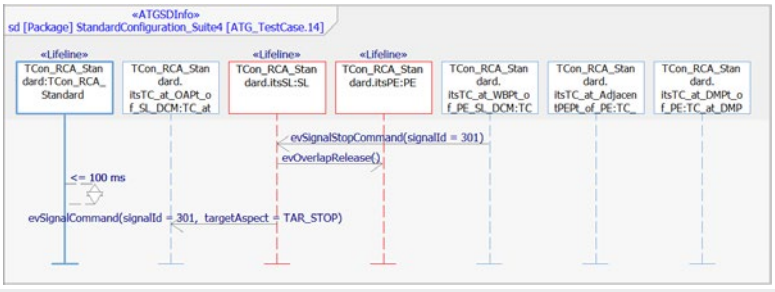
Beispielanforderung

(Safety Logic) Begrenzung der Laufzeit vom Eingang eines Kommandos, das ein Auf-Halt-Stellen eines Signals bewirkt, bis zur Ausgabe des Halt-Kommandos an Object Aggregation (unabhängig vom Ursprung

[2] Durch Umgebungstimer (s. nächster Abschnitt) besteht die Möglichkeit, auch ein verzögertes Senden zu modellieren.



4: Ausschnitt aus einem generierten Testfall, der sowohl ein 5000 ms Zeitintervall aufgrund eines Umgebungstimers als auch Outputs des SUT-Bestandteils Plan Execution mit dem Schlüsselwort „_NOT_DURING_" für die zeitliche Begrenzung eines Negativtests enthält. Das Intervall vor dem Senden der Endposition zweier Weichen bewirkt bei der Testausführung die Berücksichtigung der Weichenumlaufdauer durch den Testtreiber (zweite Beispielanforderung aus Abschnitt 2.2); die ersten beiden „_NOT_DURING_"-Nachrichten bewirken jeweils den Start einer 10.000 ms langen Überwachung auf unerwünschte weitere Anfragen der Weichen 202 und 201 durch Plan Execution, die letzten beiden jeweils den Abbruch dieser Überwachung, da der Fahrweg fertig eingestellt wurde (Beispielanforderung aus Abschnitt 2.3) Quelle: DLR



5: Ausschnitt aus einem generierten Testfall, in dem durch händische Nachbearbeitung eine maximale Systemlaufzeit durch ein 100 ms Zeitintervall spezifiziert wurde. Es betrifft die Zeitspanne vom Eingang des Nutzerbefehls bei Safety Logic, das Signal 301 auf Halt zu stellen, bis zur Ausgabe des entsprechenden Signalkommandos durch Safety Logic (Beispielanforderung aus Abschnitt 2.4) Quelle: DLR

des eingehenden Kommandos wie einem Signalhaltfall durch einen Zug via Object Aggregation, dem Abbruch der Anfrage eines Fahrtbegriffs durch Plan Execution oder einem Nutzerbefehl per Work Bench).

Umsetzung

Im Modell wird eine Time-out-Transition mit Parameter x in den entsprechenden Verarbeitungspfad zwischen SUT-Input und SUT-Output eingebaut, was zur Generierung eines Zeitintervalls von >=x ms zwischen In- und Output im Testfall führt. Anschließend wird das Relationszeichen händisch von >= auf <= in den generierten Testfällen angepasst (s. Bild 5), in denen der Verarbeitungspfad vorkommt.

Diskussion

Der genutzte Testfallgenerator unterstützt keine direkte Generierung von Constraints, die die Zeit zwischen zwei Testschritten begrenzen, obwohl solche Constraints in den Testfällen an sich darstellbar sind. Für Constraints über komplexere Schrittfolgen ist damit vermutlich ein händisches Nachtragen in den Testfällen der beste Weg. Bei Systemlaufzeiten sind die Constraints jedoch oft einfacherer Natur (Maximalzeit zwischen einem SUT-Input und dem daraufhin direkt erzeugten SUT-Output); hier trägt der unter Umsetzung beschriebene Workaround dazu bei, dass die nötige händische Anpassung nur noch minimal ist. Falls das Modell mehrere Timer enthält und der Testfallgenerator wie im vorliegenden Fall Rhapsody ATG diese bei parallelem Auftreten im Testfall miteinander verrechnet und somit die händische Anpassung doch wieder erschwert, kann alternativ eine Umsetzung ähnlich wie bei obiger zeitlicher Begrenzung von Negativtests helfen: Direkt nach dem SUT-Input kann der erwartete SUT-Output mit Schlüsselwort „_DURING_" und einer Angabe der maximal erlaubten Systemlaufzeit in den Testfall generiert werden. In jedem Fall ist bei der Testdurchführung zu beachten, dass die Laufzeiten der In- und Outputs zum/vom SUT von der durch die Testumgebung gemessenen Laufzeit abgezogen werden müssen, um die tatsächliche Systemlaufzeit zu erhalten.

2.5. Systemverhalten unter Last

Beispielanforderung

(Plan Execution und Safety Logic) Schnelle Abfolge von SUT-Inputs zu sich dem Stellbereich nähernden Zügen, zu geplanten Fahrwegen, zu neuen Zugpositionen oder

zu auf Halt zu stellenden Signalen; diese sind durch das SUT (ggf. innerhalb einer definierten Zeitspanne) korrekt abzuarbeiten.

Umsetzung

Eine geeignete Systemkonfiguration, die Serien gleichartiger SUT-Inputs (z. B. für mehrere Züge in ähnlicher Situation) erlaubt, wird erstellt. Der Ablauf des Lasttests inklusive zeitlicher Steuerung wird als Teil des Umgebungsmodells direkt modelliert (Lastteststeuerung) und der komplette Ablauf kann durch einen einzigen, speziellen Input des Testfallgenerators in den Testfall generiert werden (s. Bild 6). Die SUT-Outputs kann der Generator wie auch bei den anderen Testarten dem modellierten SUT-Verhalten entnehmen.

Diskussion

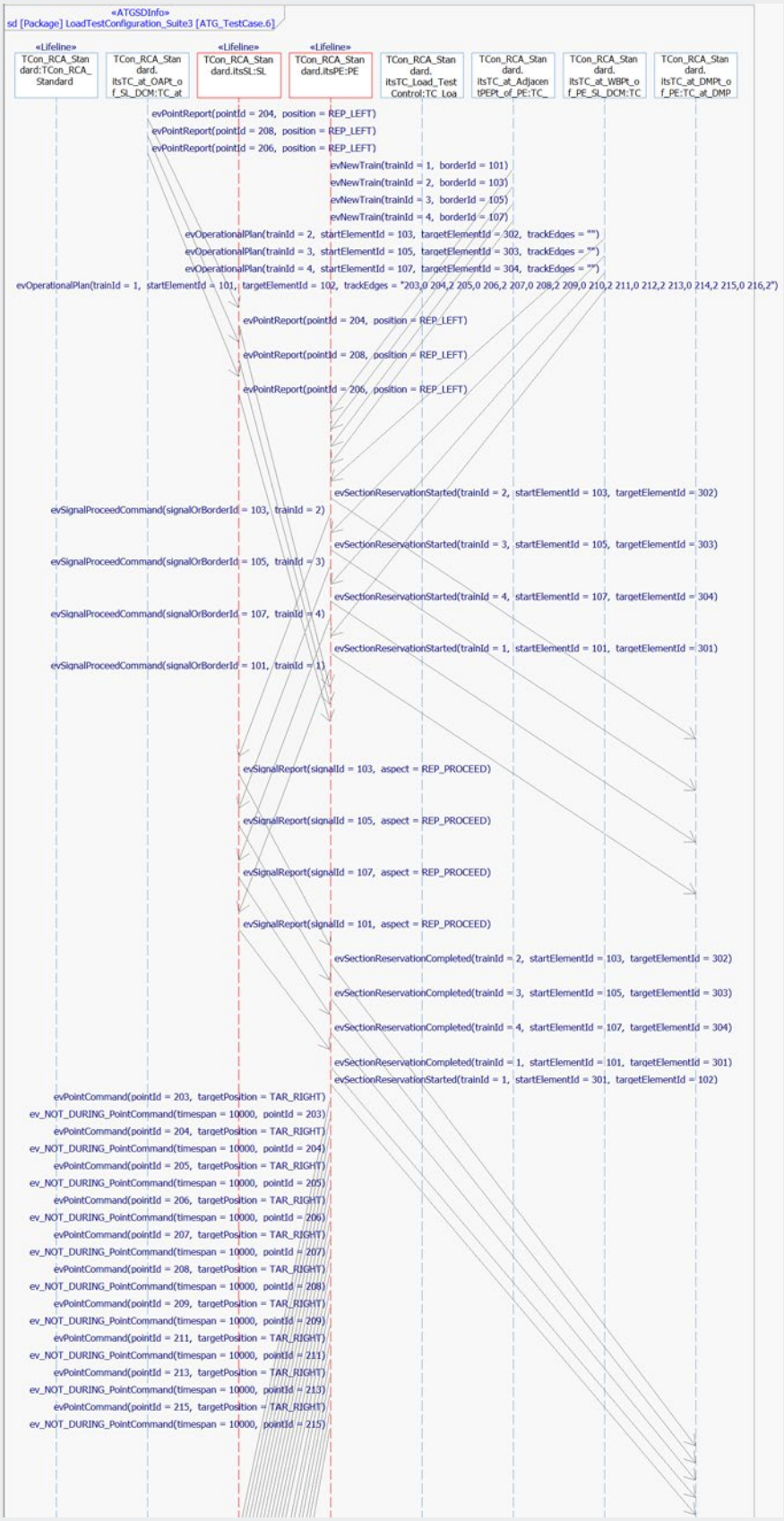
Von selbst ein sinnvolles komplexes Lasttestszenario (SUT-Inputs und deren Timing) zu finden, würde Testfallgeneratoren, die nicht darauf spezialisiert sind, überfordern. Daher ist die direkte Modellierung des Lasttests sinnvoll. Dennoch lassen sich Modell und Generator dafür nutzen, das Szenario um die erwarteten SUT-Outputs zu ergänzen. Ob die Outputs bei der Testdurchführung vollständig und ggf. auch rechtzeitig (vor den nächsten Inputs) beobachtet werden können, entscheidet über den Erfolg des Lasttests. Alternativ kann eine Generierung von expliziten zeitlichen Forderungen an die Outputs (maximale Zeitspanne) ähnlich wie für die maximalen Systemlaufzeiten (siehe Abschnitt 2.4) erreicht werden.

2.6. Ermittlung der Systemkapazität

Um die Systemkapazität zu ermitteln, bedarf es parametrisierbarer Lasttests. Entsprechend gelten die Ausführungen aus dem vorhergehenden Abschnitt 2.5 auch für Tests der Systemkapazität, wobei das Lasttestszenario nun parametrisch in der Anzahl gleichartiger SUT-Inputs und/oder ihrer zeitlichen Steuerung ist. Beispielsweise könnte das Testszenario aus Bild 6 statt für vier Züge für eine höhere Anzahl an Zügen generiert werden. Für eine Generierung einer Reihe von Testfällen zur Annäherung an die Kapazitätsgrenze des Systems wird der spezielle Input des Testgenerators entsprechend ebenso parametrisiert.

3. Modellierung von LST gemäß RCA

Die RCA befindet sich aktuell noch in der Entwicklung. Eine grundlegende Architek-



6: Ausschnitt aus einem generierten Lasttest. In schneller Abfolge – in diesem Beispielausschnitt ohne explizite Verzögerungsintervalle, d. h. so schnell wie möglich – wird das SUT mit Serien gleichartiger Inputs stimuliert: zunächst mit drei Weichenpositionen von Object Aggregation, dann mit vier Meldungen neuer Züge von Adjacent Plan Execution, und schließlich mit vier geplanten Fahrwegen für diese Züge von Work Bench. Dies führt zu einer hohen Last des SUT und einer längeren Serie von Nachrichten als Reaktionen des SUT Quelle: DLR

tur (vgl. [15] mit der in Bild 1 dargestellten Modellarchitektur) sowie grundlegende Konzepte (insbesondere DPS und Operational Plan, vgl. [16]) waren zum Zeitpunkt der Modellerstellung jedoch bereits erarbeitet und konnten in Form des RCA Baseline 0 Release 1 genutzt werden.

Eine Modellierung eines Systems unter Nutzung der Kernkomponenten und -konzepte der Architektur zu einem frühen Entwicklungszeitpunkt wie im vorliegenden Fall kann immer auch als Verifikation der Architektur verstanden werden. Insbesondere bestätigt das entstandene Modell, dass die RCA rückwärtskompatibel ist: Sie erlaubte die Modellierung eines klassischen blockbasierten LST-Systems mit Signalen, ohne dass die Architekturvorgaben als künstlich oder Erschwernis empfunden wurden. Anders als bei klassischen Stellwerken wurde jedoch keine Einschränkung auf vorkonfigurierte Fahrstraßen modelliert und somit ein allgemeinerer Fall in der RCA abgebildet (eine solche Einschränkung ließe sich bei Bedarf relativ einfach ergänzen).

Folgende durch die Nutzung der RCA hervorgerufene Besonderheiten wurden bei der Modellerstellung beobachtet:

- Die Trennung des klassischen Stellwerkskerns in die Komponenten Plan Execution und Safety Logic ist im Sinne einer Separation sicherheitsrelevanter Funktionen absolut sinnvoll. Sie hatte jedoch im Modell zur Folge, dass beide Komponenten umfangreiche und ähnliche Konfigurationsdaten benötigten. Die zentrale Bereitstellung dieser Daten durch die Device and Config Management Komponente (vgl. Bild 1) kann diesen möglichen Nachteil jedoch kompensieren.
- Die Architektur lässt (vorerst) offen, wie viel Intelligenz die Plan Execution Komponente besitzen soll. Dies betrifft z. B. eine mögliche Optimierung der Reihenfolge von Fahrwegreservierungen für verschiedene Züge oder eine mögliche Deadlockprüfung vor der Reservierung.
- Die Nutzung von DPS als abstraktem lokalem Konzept für die Sicherung von Zugfahrten ist gut möglich und sicherlich vorteilhaft für eine einheitliche Modellierung bzw. Implementierung von Sicherheitslogiken. Die konkrete Definition der Zustände einer DPS sowie die Ausgestaltung der Anwendung im Zusammenhang mit verschiedenen Infrastrukturelementen lässt RCA jedoch (bislang) offen, obwohl hier mutmaßlich immer wieder ähnliche Modelle / Implementierungen entstehen dürften.
- Die durch RCA definierten Architekturebenen [15] der Object Aggregation sowie der Device Abstraction zwischen der Safety Control und der Device Control Ebene scheinen nicht besonders viel Funktionalität zu umfassen; zumindest war im erstellten Modell, das mit stark vereinfachten Nachrichtenformaten zwischen RCA-Komponenten arbeitet, keine explizite Modellierung nötig bzw. eine simple Weiterleitung der Nachrichten ausreichend.

4. Fazit

Während viele systemmodellbasierte Testfallgeneratoren mit grundlegenden zeitbezogenen Anforderungen in der Form modellierter System-Timer umgehen können und diese als Verzögerungen in den Testfällen abbilden können, ist dies bei weiteren Arten zeitbezogener Anforderungen wie z. B. maximalen Systemlaufzeiten oft nicht mehr der Fall. Andererseits kann es effizient sein, Testfälle für solche Anforderungen möglichst zusammen mit denjenigen für alle anderen Anforderungen aus dem gleichen Modell zu generieren. Anhand von konkreten LST-Anforderungsbeispielen und deren Umsetzung in einem Modell und durch einen Testfallgenerator wurde aufgezeigt, wie verschiedene Arten zeitbezogener Anforderungen in den modellbasierten Test einbezogen werden können.

Darüber hinaus handelt es sich bei dem erstellten Modell um eine Umsetzung zentraler Bestandteile der aktuell in Entwicklung befindlichen Reference CCS Architecture. Somit konnte die Kompatibilität der RCA mit der gewählten klassischen blockbasierten LST mit Signalen gezeigt werden. Die knapp berichteten Erfahrungen mit Besonderheiten bei der Anwendung der RCA können hilfreich für künftige RCA-Anwender sowie die weitere Entwicklung der RCA sein. ●

Literatur

[1] Kramer, A.; Legeard, B.: 2019 Model-based Testing User Survey: Results, 2020.
[2] DIN EN 50126-1:2018-10 Bahnanwendungen - Spezifikation und Nachweis von Zuverlässigkeit, Verfügbarkeit, Instandhaltbarkeit und Sicherheit (RAMS) - Teil 1: Generischer RAMS-Prozess; Deutsche Fassung EN 50126-1:2017.
[3] DIN EN 50128:2012-03 Bahnanwendungen - Telekommunikationstechnik, Signaltechnik und Datenverarbeitungssysteme - Software für Eisenbahnsteuerungs- und Überwachungssysteme; Deutsche Fassung EN 50128:2011.
[4] DIN EN 50129:2019-06 Bahnanwendungen - Telekommunikationstechnik, Signaltechnik und Datenverarbeitungssysteme - Sicherheitsbezogene elektronische Systeme für Signaltechnik; Deutsche Fassung EN 50129:2018 + AC:2019.
[5] D. Schwencke (2022): Testfallgenerierungsgestützte Validierung und Verifikation von EULYNX-Spezifikationen. SIGNAL+DRAHT (114) 6/2022, S. 43-53.
[6] EULYNX: Requirements specification for subsystem Point. Eu.Doc.36, baseline 3.1, Jun 19, 2020.
[7] IBM Rhapsody ATG User Guide Website, https://www.ibm.com/docs/en/rhapsody/9.0.1?topic=atg-automatic-test-generation-user-guide, abgerufen am 31.08.2022, 15:05.
[8] J. Peleska, E. Vorobev, F. Lapschies, C. Zahlten (2011): Automated Model-Based Testing with RT-Tester. Technical Report, 2011-05-25.
[9] Conformiq Designer Website, https://www.conformiq.com/products/conformiq-designer/, abgerufen am 05.09.2022 16:33.
[10] K. Berkenkötter, R. Kirner (2005): Real-time and hybrid systems testing. In M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, A. Pretschner (eds.): Model-based Testing of Reactive Systems (Lecture Notes in Computer Science, vol. 3472), Springer Berlin, 355–387.
[11] RCA Published Releases Website, https://public.3.basecamp.com/p/KeehzqFmXv5R2N7tGDjaEokq, abgerufen am 05.09.2022 16:37.
[12] IBM Rhapsody Website, https://www.ibm.com/products/systems-design-rhapsody, abgerufen am 31.08.2022 14:58.
[13] OMG SysML Website, https://www.omg.org/spec/SysML/, abgerufen am 31.08.2022, 14:56.
[14] EUG RCA Website, https://ertms.be/workgroups/ccs_architecture, abgerufen am 31.08.2022 14:30.
[15] Reference CCS Architecture Initiative (2021): RCA Architecture Poster – Preliminary issue. RCA.Doc.40, RCA Baseline 0, Version 0.2 (0.B). Verfügbar unter [11].
[16] Reference CCS Architecture Initiative (2021): RCA Domain Knowledge – Preliminary issue. RCA.Doc.18, RCA Baseline 0, Version 0.2 (0.A). Verfügbar unter [11].

Summary

Model-based testing of time-related signalling system requirements taking the RCA into account

The automatic generation of test cases from behavioural system models can increase the efficiency and quality of the design of tests of standard functional requirements. By means of a concrete model and test case generator it is demonstrated how to furthermore generate also tests of different typical time-related requirements on signalling systems. In addition, experiences with modelling according to the "Reference CCS Architecture" made in this context are reported.