Automatisiertes Testen von verteilten Systemen über Petrinetze

Dipl.-Ing. T. Ruß*, tim.russ@ifak.eu
Dipl.-Ing. S. Magnus*, stephan.magnus@ifak.eu
Dr.-Ing. J. Krause*, jan.krause@ifak.eu
* Institut für Automation und Kommunikation e. V., Werner-Heisenberg-Straße 1, D-39106 Magdeburg

Zusammenfassung: Dieser Beitrag stellt eine Methode vor, um mittels modellbasierter Tests ein verteiltes System über seine Netzwerk-Kommunikation zu validieren. Diese wird kontextabhängig gegen ein Sollverhalten geprüft und ggf. manipuliert. Die Beschreibung des Sollverhaltens erfolgt hierbei durch eine sequenzbasierte Notation. Dadurch ist es einfach und intuitiv möglich, Testszenarien zu implementieren und automatisiert durchzuführen. Zur Laufzeit wird das Netzwerkverhalten über ein Petrinetz effizient validiert. Außerdem ist es möglich, ein entsprechendes Gerät in ein Netzwerk zu integrieren und den Datenverkehr auf vielfältige Weise zu manipulieren.

Schlüsselwörter: Modellbasierter Test, Netzwerke, PROFINET, Car2X, Petrinetze, Sequenzdiagramme

1. Motivation und Ansatz

Ein weit verbreitetes Modell der Software-Entwicklung ist das V-Modell. Hier wird deutlich, wie in jeder Abstraktionsebene Tests genutzt werden, um Fehler aufzudecken und die Entwicklungen jeder Ebene abzusichern. Beim Systemtest wird die Produktivumgebung nachgestellt (*System under Test*, SUT) und in ihrer Gesamtheit getestet. Bei verteilten Systemen wie Automatisierungsanlagen werden die einzelnen Geräte über ein Netzwerk bzw. Bussystem zusammengeschaltet. Für Ethernet existieren weit verbreitete Protokolle wie PROFINET oder EtherCAT. In den Geräten arbeiten Automatenmodelle, um die Protokollstacks zu realisieren. Diese sind bekannt und können applikativ direkt auf den Geräten für Tests in einen definierten Zustand versetzt und anschließend ihre Ausführung validiert werden. Dieses Vorgehen wurde zum Beispiel in [1] gezeigt.

Zusätzlich ist es üblich, die resultierende Netzwerk-Kommunikation des Ethernet-Netzwerkes mitzuschneiden und auf Korrektheit zu prüfen. Diese Kommunikation ist das von außen beobachtbare Resultat der in den Geräten aktiven Automatenmodelle, obgleich es nicht alles über diese inneren Zustände verrät. Nichtsdestotrotz ist der Systemtest über die Netzwerkschnittstelle ein gutes Mittel, die gesamtheitliche Validierung zu unterstützen. Hierfür erfordert es effiziente Testmethoden. Noch ist die manuelle, passive Prüfung über Wireshark weit verbreitet, da die Software frei und einfach anzuwenden ist.

Für eine ausreichend hohe Testabdeckung und weitreichende Testautomatisierung bietet es sich jedoch an, eine systematische Methodik auf Basis eines modellbasierten Tests zu finden. Hier wurden am ifak erfolgreich Arbeiten durchgeführt und werden im Folgenden vorgestellt. Die Idee ist, verteilte Systeme verschiedener Domänen über die beobachtbare Netzwerkkommunikation zu testen. Abbildung 1 zeigt das prinzipielle Vorgehen. Mitgeschnittene Nachrichten (Ist-Verhalten) werden automatisch gegen Nutzervorgaben (Soll-Verhalten) geprüft, um frühzeitig Fehler aufzudecken. Dies wird derzeit für Tests im Umfeld von PROFINET-Anlagen (Ethernet) und Car2X/V2X (IEEE 802.11p/WLANp, siehe z. B. [2]) angewendet.

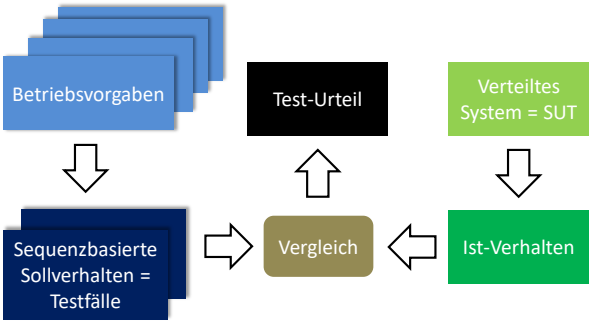

Abbildung 1: Ablauf des modellbasierten Tests für verteilte Systeme

Der Ansatz ist, den Nutzer über eine domänenspezifische Sprache in Form von Sequenzdiagrammen diverse Sollverhalten für den Netzwerkverkehr beschreiben zu lassen, welche einzelne Testfälle abstrahieren und nachbilden. Um das Systemverhalten zur Laufzeit nachverfolgen zu können, werden die Testfälle automatisch in Petrinetze umgewandelt. Über das Markenspiel lässt sich feststellen, ob das Sollverhalten erfüllt oder verletzt wurde. Dies hilft, fehlerhafte Testfälle zu identifizieren, um nur bei diesen manuell nach dem zugrunde liegenden Fehler suchen zu müssen. Dies spart Zeit und letztendlich Kosten.

Auf Basis der entwickelten Konzepte wurde bereits eine performante Software auf Basis von C++ implementiert, die derzeit für Tests von PROFINET- sowie Car2X/WLANp-Systemen eingesetzt wird. Hauptsächlicher Testgegenstand sind die Daten der höheren Netzwerk-Layer, vom Protokoll (IP, UDP, TCP, DCE/RPC…) bis hin zu den Applikationen und Nutzerdaten.

2. Sollverhalten und Testfälle über sequenzbasierte Notation

Bei Systemtests über die Netzwerk-Schnittstelle liegen einzelne Sollverhalten in Dokumentationen oder beim Anwender als Expertenwissen vor. Sie beschreiben Abläufe, die sich aus den verwendeten Protokollstacks oder

individuellen Applikationen ergeben. So findet zwischen PROFINET-Steuerungen und –Feldgeräten stets ein Verbindungsaufbau statt, bevor Nutzdaten übertragen werden können. Hier würde ein Tester jedes Mal nachsehen müssen, ob die Verbindung erfolgreich aufgebaut wurde, indem er die obligatorischen Netzwerk-Nachrichten in einem Mitschnitt identifiziert. Oder aber, es wird über eine Car2X-Nachricht eine Warnung vor einem Stauende als Broadcast an alle KFZ in der Nähe übertragen. Dabei könnte ein Tester prüfen, ob Parameter wie die standardisierte ID der Warnung korrekt sind.

Im Regelfall findet eine Interaktion zwischen Kommunikationspartnern statt. Diese lässt sich über Sequenzdiagramme darstellen. Für diese sind die Standards *UML 2.X Sequence Diagram* der ISO und Object Management Group (siehe z. B. [3] und [4] sowie *MSC* der ITU-T (siehe [5]) verfügbar.

Die erste Innovation besteht nun darin, dass am ifak eine domänenspezifische Sprache geschaffen wurde, um Sollverhalten über solche Sequenzen möglichst einfach notieren zu können, maschinen-lesbar zu machen und automatisch gegen mitgeschnittene Datenpakete zu prüfen. Die Syntax folgt dem Muster: *Nachrichtenname(Sender->Empfänger)*. Welche Parameter sich hinter dem Nachrichtennamen sowie Sender und Empfänger verbergen, lässt sich aus Definitionen importieren. Neben einfachen Sequenzen wurden Fragmente aus dem UML 2.X SD-Standard implementiert. Sie erlauben die Beschreibung von Schleifen (wie zyklischer Datenabfragen) oder parallelen Abläufen (wie parallelen Verbindungsaufbauten).

Derart notierte Sollverhalten beschreiben in der Regel das Gutverhalten bei Hochlauf und Dauerbetrieb eines Systems. Über UML-Fragmente wie die Negation (neg) lassen sich darüber hinaus unzulässige Abläufe notieren, um Fehlverhalten aufdecken zu können. Letztendlich repräsentiert ein Soll- oder Fehlverhalten einen Testfall, der beim Prüfen gegen Realdaten erfolgreich sein muss, um in einem Testreport (Liste aus absolvierten Testfällen) als „OK“ eingestuft zu werden.

Eine weitere Innovation ist die Realisierung von Manipulationsmöglichkeiten. In die domänenspezifische Sequenzsprache wurden Aktionen eingebaut, die vom Anwender kontextabhängig verwendet werden können. Dazu zählen bisher das Löschen von Nachrichten sowie das gezielte Ändern von Parametern. Nachrichten lassen sich Löschen, indem ein Gerät mit der vorgestellten Testsoftware als Man-in-the-middle mit zwei Schnittstellen in ein Ethernet-Netzwerk eingebracht wird. So bietet es sich bei PROFINET-Netzwerken an, die Leitung direkt vor einem Controller aufzutrennen. Hier leitet die Test-Software zum Löschen markierte Nachrichten einfach nicht an die komplementäre Schnittstelle weiter.

Durch diese Manipulationen ist es möglich, gezielt Fehlverhalten von Geräten nachzustellen, um beispielsweise Protokollstacks auf korrekte Reaktionen zu prüfen. Die Prüfung der erwarteten Nachrichten wie nachfragende Pings nach einer Nachrichten-Löschung kann im gleichen Sollverhalten erfolgen, in dem die Manipulation beschrieben ist.

3. Validierung zur Laufzeit über ein Petrinetz

Die verfügbaren Sequenzdiagramm-Standards sind dazu konzipiert, generelles Verhalten und Interaktionen zu modellieren. Sie besitzen jedoch keine Möglichkeit, Eingabedaten zu erhalten und einen Interaktionsverlauf zu speichern. Dadurch erlauben diese Modelle keine Aussage darüber, welche Interaktionen bzw. Ereignisse als nächstes auftreten können bzw. bereits aufgetreten sind. Dies ist hingegen über Modelle gerichteter Graphen wie Endliche Zustandsautomaten (Finite State Machines, FSM) und Petrinetze möglich. Ein Automatenmodell erlaubt lediglich einen aktiven Zustand zur gleichen Zeit. Diese Betrachtungsweise ist angebracht, wenn man bspw. den Zustand eines Protokollstacks auf einem PROFINET-Gerät betrachtet. Ein Netzwerk kann jedoch von mehreren Kommunikationsbeziehungen und Applikationen genutzt werden, deren Datenpakete in zuvor nicht absehbarer Reihenfolge übertragen werden. Falls ein Anwender hier parallele Abläufe beschreiben und testen will, bietet sich ein Petrinetz als Erweiterung des Automatenmodells an. Es besteht aus Stellen (passive Elemente), Marken auf Stellen zur Repräsentation verfügbarer Ressourcen bzw. Aktionen, Transitionen (aktive Elemente) zur Repräsentationen von Ereignissen sowie verbindenden Kanten.

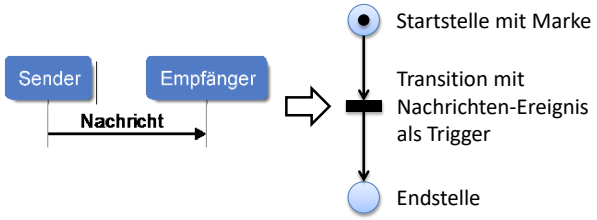

Abbildung 2: Überführung eines Sequenzdiagramms in ein Petrinetz

Für die hier vorgestellte Testmethodik wurde die automatische Überführung der Sequenz-basierten Testnotation in ein entsprechendes Petrinetz untersucht und softwaretechnisch umgesetzt. Abbildung 2 zeigt ein einfaches Beispiel. Jede Nachricht des Sollverhaltens wird in eine Transition überführt. Wird die entsprechende Nachricht mitgeschnitten und erfolgreich identifiziert, so kann die Transition feuern, Marken bewegen und im Petrinetz den Verlauf des Testfalls vorantreiben.

Solche Überführungen von Sequenzdiagrammen in Petrinetzen wurden in der

Literatur bereits untersucht (siehe z. B. [6] und [7], hier aber angepasst und erweitert. So wurde sich zunächst dafür entschieden, die Kommunikationspartner selbst nicht über zusätzliche Stellen zu modellieren. Ihre Parameter (wie IP-Adresse) werden der Nachricht und damit einer Transition zugeordnet. Dies liegt darin begründet, dass durch passives Mitschneiden von Nachrichten im Netzwerk nur das Sende-, nicht aber das Empfangsereignis detektiert werden kann. Letzteres wird daher auch nicht mitmodelliert.

Weiterhin wurde für dieses Testvorgehen festgelegt, dass jedes Petrinetz eine globale Start- sowie Endstelle besitze. Bei der Initialisierung eines Testfalls wird eine Marke auf die Startstelle gelegt. Die daran anschließenden Transitionen werden dadurch befähigt, als erstes zu feuern. Wird durch den Testverlauf eine Marke auf die Endstelle gelegt, so gilt der Testfall als erfolgreich beendet.

Wurde vom Nutzer eine Manipulation im Sollverhalten vorgesehen, so wird diese als Aktion an eine Transition gehängt. Sie wird ausgeführt, wenn die Transition feuert.

4. Erweiterung des Test-Adapters für vielfältige Ereignisse aus dem System under Test

Der erste Ansatz für dieses Testvorgehen berücksichtigte lediglich Nachrichten, die über ein Netzwerk wie Ethernet oder WLAN übertragen werden. Ein Adapter wurde geschaffen, der folgende Umwandlungen vornimmt: Beim Testbeginn werden alle Nachrichten des Sollverhaltens in einer Liste gesammelt. Für jeden Eintrag wird ein abstraktes Ereignis und entsprechend eine Transition im Petrinetzmodell erstellt. Danach werden über eine PCAP-Schnittstelle kontinuierlich alle eingehenden Datenpakete des zu testenden Systems erfasst. In den meisten Fällen entsprechen diese direkt Nachrichten. Durch Überschreiten von Paketgrößen kann es jedoch vorkommen, dass Nachrichten in mehrere Pakete segmentiert werden. In diesen Fällen setzt der Adapter mehrere Pakete zu einem Nachrichtenobjekt zusammen. Es erfolgt eine Dekodierung der verwendeten Protokollschichten, zum Beispiel Ethernet → IP → UDP → DCE/RPC → PNIO_CM. Es entsteht ein Nachrichten-Objekt, in dem auf alle enthaltenen Parameterwerte zugegriffen werden kann. Durch einen Parametervergleich wird jedes Mal geprüft, ob solch eine Nachricht in der Liste der erwarteten Nachrichten auftaucht. Falls ja, so wird für sie das zugehörige Ereignis herausgesucht und versucht, die zugehörige Transition feuern zu lassen. Dadurch ist das Petrinetz unabhängig von der Domäne. Die Transformations- und Ausführungs-Algorithmen können beispielsweise sowohl für PROFINET als auch Car2X eingesetzt werden. Die Interpretation der Nachrichten zu abstrakten Ereignissen geschieht ausschließlich im Adapter.

Mittlerweile haben sich Anwendungsfälle ergeben, in denen es nicht mehr genügt, nur Sollverhalten von Netzwerknachrichten zu testen. Daher wird derzeit die Erweiterung des Konzepts auf digitale Signale erforscht. Beispielsweise kann erforderlich sein, in einem Szenario kooperativer Verkehrssysteme zu testen, ob und wann eine straßenseitige Car2X-Einheit (Roadside Unit) eine Car2X-Nachricht verschickt. Über Schnittstellen wie GPIOs (General Purpose Input/Output Pins) an der Roadside Unit lassen sich interne Ereignisse über digitale Pegelwechsel nach außen und dem Gerät mit der Testsoftware zuführen. Zusätzlich lässt sich der Versand der Car2X-Nachricht über einen in Funkreichweite installierten Drahtlosrouter mitschneiden.

Der Adapter wird nun so erweitert, dass neben dem Mitschneiden, Dekodieren und Interpretieren von Netzwerknachrichten auch digitale Pegelwechsel über GPIOs erfassbar sind. Auch sie werden einem abstrakten Ereignis und letztendlich einer Transition im Petrinetz zugeordnet.

5. Darstellung an einem Anwendungsfall

Als Anschauungs-Beispiel soll eine einfache PROFINET-Anlage aus einer Steuerung (Controller) und einem Feldgerät (Device) dienen. Die verbindende Ethernet-Leitung wird aufgetrennt und das Testsystem (PC mit Testsoftware) über zwei Netzwerkkarten eingebracht.

Nach dem Start eines Tests werden alle Netzwerkpakete standardmäßig weitergeleitet, sodass das Testsystem für das System under Test nicht sichtbar ist.

Ein Testziel sei nun, die Kommunikation gezielt zu stören, um die korrekte Reaktion der beteiligten Geräte zu prüfen. So wird der Controller nach erfolgreicher Entdeckung des Devices versuchen, eine Verbindung aufzubauen. Die erste obligatorische Nachricht ist ein Connect Request. Dieses wird im sequenzbasierten Sollverhalten notiert. Jedoch wird zusätzlich eine Aktion zum Löschen angehängt. Dies bedeutet: Als erstes ist diese Nachricht (von diesem Sender zu diesem Empfänger) zu erkennen. Danach ist sie nicht weiterzuleiten. Der Anwender weiß, dass der Controller keine Antwort erhalten und mit einem Ping nachfragen wird. Zur Veranschaulichung der Möglichkeiten werde zusätzlich festgelegt, dass auch der Ping zu löschen ist. Über ein Loop-Fragment (dem UML SD-Standard entlehnt) lässt sich dies beliebig oft wiederholen. Im Anschluss lässt sich das folgende, korrekte Verhalten notieren, nämlich ein erneutes Connect Request gefolgt von einem erfolgreichen Connect Response des Devices. Abbildung 3 verdeutlicht das Vorgehen. Letztendlich vereint dieser

Testfall die Prüfung von Gutverhalten, kontextuelle Manipulation und die Reaktion darauf.

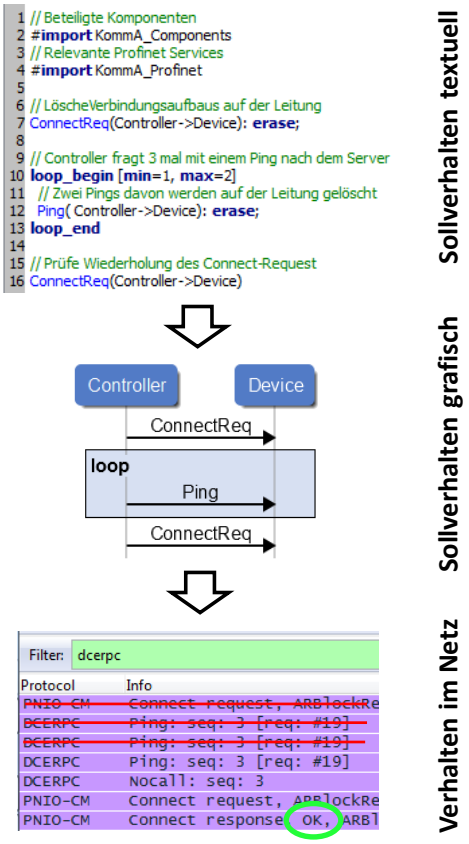

Abbildung 3: Nutzen der "erase"-Aktion, um spezifische Ethernet-Nachrichten zu löschen

6. Grenzen und Erweiterungen

Nach wie vor besteht die Herausforderung, die Testfälle bzw. Sollverhalten zu erstellen. Prinzipiell ist für solche Tests eine ausreichend hohe Testabdeckung erforderlich. Aus den in den Kommunikationspartnern ablaufenden Applikationen und Automatenmodellen resultierenden jedoch unzählige von außen beobachtbare Ereignisse, sodass man von einer Zustandsexplosion sprechen kann. Die Abbildung aller möglichen Abläufe über ein Modell ist praktisch nicht möglich. Daher ist stets eine Auswahl an Testfällen zu treffen, um eine ausreichende Testtiefe zu erreichen. Das hier vorgestellte Vorgehen soll bisher manuelle Netzwerktests automatisieren und applikative Tests ergänzen. Hier existieren meist nur eine Handvoll Testfälle, die sich leicht durch die vorgestellte Methodik adaptieren lassen.

Im Rahmen von Tests werden einzelne Testfälle nacheinander ausgeführt. Dabei wird das System under Test jedes Mal in einen definierten Ausgangszustand versetzt, zum Beispiel den Zeitpunkt vor dem Hochfahren einer Automatisierungsanlage. Dies resultiert für die vorgestellte Methodik darin, dass stets nur eine Sollsequenz mit einem daraus generierten Petrinetz gegen Ist-Daten zur gleichen Zeit geprüft wird.

Hingegen werden Diagnosen eingesetzt, um im Betrieb befindliche Systeme auf Fehler und deren Ursachen hin zu untersuchen, zum Beispiel nach dem Hochlauf einer Anlage. Im Unterschied zu Tests ist es erforderlich, dass mehrere Aspekte bzw. Sollverhalten gleichzeitig gültig sind. Ein Diagnosesystem muss sie allesamt auswerten, ohne das System under Test jedes Mal in einen vordefinierten Zustand zurück zu versetzen.

Derzeit wird untersucht, wie sich die Petrinetz-Methodik auf Diagnosen erweitern lässt. Die Idee ist, Betriebsvorgaben und System-Sollverhalten (Protokollstacks, Applikationen…) über gleichzeitig gültige Regeln zu definieren. Es liegt nahe, für jedes Sollverhalten ein eigenes Petrinetz zu erstellen, welche alle gleichzeitig zur Laufzeit ausgeführt werden. Neben diesen verhaltensbasierten Vorgaben (Verbindungsaufbau, zyklische Datenverkehr etc.) werden aber mehr Typen von Regeln nötig sein, unter anderem zur Beschreibung von Strukturen (bspw. Netzwerk-Topologien) und Funktionen. Die Herausforderung besteht nun darin, eine Notation für diese verschiedenen Typen zu schaffen, aus der sich vergleichbare Petrinetze erstellen lassen.

7. Literaturverzeichnis

[1] S. Magnus, T. Ruß und J. Krause, „Anforderungs- und modellbasierte Testfallgenerierung und Testdurchführung unter Nutzung von Methoden zur Netzwerkanalyse," Magdeburg, 2016.

[2] CAR 2 CAR Communication Consortium, „CAR 2 CAR Communication Consortium: Mission & Objectives," [Online]. Available: https://www.car-2-car.org. [Zugriff am 22 Juli 2016].

[3] C. Rupp, S. Queins und B. Zengler, UML 2 glasklar, München, Wien: Carl Hanser Verlag, 2007.

[4] Object Management Group, „OMG Unified Modeling Language (OMG, UML), Infrastructure. Version 2.4.1.," 5 August 2011. [Online]. Available: http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF.

[5] ITU-T, „Message Sequence Chart (MSC). ITU-T Recommendation Z.120.," 2011.

[6] J. M. Fernandes und Ó. R. Ribeiro, „Some Rules to Transform Sequence Diagrams into Coloured Petri Nets," Dept. of Informatics, University of Minho, Portugal, 2006.

[7] O. Kluge, „Petri nets as a Semantic Model for Message Sequence Chart Specifications," citeseerx.ist.psu.edu, 2001.