Bringing Model Checking Closer To Practical Software Engineering

Remenska, D.

2016

document version
Publisher's PDF, also known as Version of record

Doordat software voortdurend groeit in grootte en complexiteit is het een steeds grotere uitdaging om er voor te zorgen dat het zich correct gedraagt. Dit geldt vooral voor gedistribueerde systemen waar een veelheid aan componenten gelijktijdig draaien. Hierdoor is het moeilijk om al het mogelijke gedrag van het systeem als geheel te anticiperen. Sommige ontwerpfouten, zoals impasses en race-condities, blijven vaak onopgemerkt als het testen van het software de enige vorm van controle is tijdens de ontwerpcyclus van het software. Zelfs als fouten ontdekt worden tijdens het draaien van het software dan is het vaak tijdrovend (of zelfs onmogelijk) om de uiteindelijke oorzaak te achterhalen doordat de programmeur nauwelijks invloed heeft op de uitvoering van onderdelen die gelijktijdig draaien en door de hoeveelheid mogelijke scenario's die het probleem veroorzaakt kunnen hebben. Dit is vooral waar voor grootschalig gedistribueerde systemen zoals het Worldwide Large Hadron Collider Computing Grid (Wereldwijd Grote Hadronen-Botser Computernetwerk).

Formele verificatie methoden bieden grondiger manieren om te bepalen of een systeem voldoet aan bepaalde vereisten met betrekking tot gedrag. Een erg effectieve verificatie techniek is modelchecken. Modelchecken is een op wiskunde gebaseerd algoritmische procedure die tegenwoordig automatische gedaan wordt door vele actief onderhouden en uitontwikkelde gereedschappen. Een nadeel is echter dat het nodig is om bedreven te zijn in formele-taalnotaties zoals procesalgebra en temporele logica die onontbeerlijk zijn voor zowel de beschrijving van het model van het systeem en de eisen aan het gedrag die door deze gereedschappen getoetst moeten worden.

In dit proefschrift beantwoorden we de vraag hoe modelchecken geïntegreerd kan worden in de gebruikelijk ontwikkelingscyclus van distribueerde, datagedreven software's met een realistische grootte, met het automatiseren van die aspecten die gespecialiseerde kennis van formele methoden vereisen. Voor dit doel kozen we een formalisme dat krachtig genoeg is om ontwerpfouten die regelmatig voorkomen in dergelijke software's te modelleren en aan te pakken. De aanleiding is DIRAC, een raamwerk voor gedistribueerde "grid" software dat wordt ontwikkeld en gebruikt door de fysicagemeenschap van de Europese organisatie voor deeltjesfysica (CERN). We onderzoeken of het mogelijk is om de mCRL2 taal te gebruiken door van twee deelsystemen van DIRAC systematisch de broncode te abstraheren en het gedrag te modelleren. De studies geven aan dat mCRL2 de nodige mechanismen heeft om van de deelsystemen de gelijktijdigheid, data-abstractie en manipulatie getrouw te modelleren. Door simulatie, visualisatie, en toetsing van de resulterende modellen met de mCRL2 gereedschapskist werd het voor ons mogelijk om een beter inzicht te krijgen in het gedrag van het systeem en het afspelen van tegenvoorbeelden hielp in de plaatsbepaling van de opgemerkte problemen. Hoewel een deel van het foutiefgedrag al in de praktijk naar voren was gekomen voordat de deelsystemen formeel gemodelleerd waren, was het nog niet gelokaliseerd binnen de tijdspanne waarin we de modellering en verificatie uitgevoerd hebben.

Het met de hand construeren van een formeel model voor een concreet software kan tijdrovend zijn. Daarbij is er het risico om fouten te maken en moet het model om nuttig te zijn bijgewerkt blijven bij voortdurende veranderingen

aan het software. Programma's bevatten te veel taalspecifieke details om als uitgangspunt te dienen voor het afleiden van modellen. Bij software engineering worden hoogniveau visuele ontwerpen gemaakt voor overleg over en het valideren van vereisten voordat de uitwerking en het testen plaats vinden. Universal Modeling Language (UML, universele modelleringstaal) wordt algemeen geaccepteerd als een visuele modelleertaal voor dit doel. We presenteren een aanpak voor het automatisch afleiden van een mCRL2 formeel model door de transformatie van UML sequentie- en activiteitsdiagrammen. We bespreken de keuzes op het gebied van semantiek die we gemaakt hebben waar het gaat om ambiguïteiten in officiële halfformele UML-semantiek en we vergelijken onze aanpak met reeds bestaande binnen het kader van een bekende classificatie. De transformatie bewaart de object georiëneerde structuur van het systeem en maakt het eenvoudig om tegenvoorbeelden van verificatie visueel te presenteren als sequentiediagrammen. Om wat praktijk ervaring en vertrouwen te wekken in de juistheid van de transformaties passen we de gereedschapondersteunde aanpak toe op het nieuwe werkbelastingsysteem van DIRAC. We ontdekken de uiteindelijke oorzaak van een logische fout die er voor zorgt dat er geen vooruitgang meer geboekt wordt. Een fout die eerder tijdens testfase van deze functie waargenomen is.

In modelchecken moeten eigenschappen van gedrag uitgedrukt worden als formules in temporele logica. De mCRL2 gereedschapskist vereist het gebruik van modale μ-calculus voor dit doel; een erg expressieve logica, maar niet erg intuïtief of toegankelijk. Vereisten voor gedrag van software worden normaal gesproken uitgedrukt in natuurlijk taal en zijn als zodanig op meerdere manieren uit te leggen. In het kader van de centrale onderzoeksvraag, het correct uitdrukken van gedragseigenschappen dichter bij programmeurs te brengen, voegen we een eigenschapassistent gereedschap met de naam PASS toe aan de op UML gebaseerde frontend van de mCRL2 gereedschappen. Dit gereedschap assisteert leken in het naar voren halen van eigenschappen die de vereiste gedrag precies en niet-ambigu uitdrukken. Het is gebaseerd op een bekende classificatie van patronen van eigenschappen die we uitgebreid hebben met nieuwe patroonvariaties voor de event-based modale μ-calculus. Naast de μ-calculus formules biedt het gereedschap een samenvatting in natuurlijke taal en een UML sequentiediagram dat de eigenschap afbeeldt. Verder genereert PASS automatisch voor een deel van de eigenschappen observatiestructuren die (potentieel) gebruikt kunnen worden voor de efficiëntere eigenschapgedreven verificatie tijdens de uitvoering. We demonstreren het gebruik van PASS met betrekking tot het gedrag van DIRAC.

De classificatie van eigenschappatronen is gebaseerd op een groot literatuuronderzoek naar het gebruik van specificatieformalismen in de praktijk. Aangezien we geen μ-calculus formules aantroffen in de verzameling waren we benieuwd of het gebruik significant afwijkt van dat van andere formalismen. We hebben 25 gepubliceerde werken in kaart gebracht die μ-calculus gebruiken om eigenschappen uit te drukken van systemen uit verschillende domeinen. Ten opzichte van de 178 eigenschappen uit het literatuuronderzoek bieden onze extra patronen een 10% extra dekking van de geclassificeerde patronen. De modale

μ-calculus, die als minder intuïtief beschouwd wordt dan meer algemene logica's (zoals LTL - Lineaire Temporele Logica, en CTL - Computationele Boom Logica), is instaat sommige patronen van eigenschappen te omvatten waartoe afzonderlijk LTL en CTL niet instaat zijn. We hebben een verdeling van de patronen waargenomen die behoorlijk consistent is met vergelijkbare onderzoeken naar andere formalismen. De resultaten geven ook aan dat het eenvoudig is om subtiele fouten te maken bij het handmatig samenstellen van temporele logicaformules.

Als een mogelijke toekomstige richting zou het interessant zijn om dit werk uit te breiden buiten het ontdekken van gedragsproblemen, in het bijzonder op het gebied van het inschatten van prestaties aan de hand van UML modellen. Hoewel dit type analyse van primair belang is voor "real-time"- en embedded systemen kunnen ook waardevolle inzichten verkregen worden over de communicatiekosten, vertragingen en verschillende flessenhalzen in gedistribueerde systemen.