Stichting NIOC en de NIOC kennisbank

Stichting NIOC (www.nioc.nl) stelt zich conform zijn statuten tot doel: het realiseren van congressen over informatica onderwijs en voorts al hetgeen met een en ander rechtstreeks of zijdelings verband houdt of daartoe bevorderlijk kan zijn, alles in de ruimste zin des woords.

De stichting NIOC neemt de archivering van de resultaten van de congressen voor zijn rekening. De website www.nioc.nl ontsluit onder "Eerdere congressen" de gearchiveerde websites van eerdere congressen. De vele afzonderlijke congresbijdragen zijn opgenomen in een kennisbank die via dezelfde website onder "NIOC kennisbank" ontsloten wordt.

Op dit moment bevat de NIOC kennisbank alle bijdragen, incl. die van het laatste congres (NIOC2025, gehouden op donderdag 27 maart 2025 jl. en georganiseerd door Hogeschool Windesheim). Bij elkaar zo'n 1500 bijdragen!

We roepen je op, na het lezen van het document dat door jou is gedownload, de auteur(s) feedback te geven. Dit kan door je te registreren als gebruiker van de NIOC kennisbank. Na registratie krijg je bericht hoe in te loggen op de NIOC kennisbank.

Het eerstvolgende NIOC vindt plaats op 18 maart 2027 in Arnhem en wordt georganiseerd door HAN University of Applied Sciences.

Reacties over de NIOC kennisbank en de inhoud daarvan kun je richten aan de beheerder:

R. Smedinga kennisbank@nioc.nl.

Vermeld bij reacties jouw naam en telefoonnummer voor nader contact.

Het methodologisch ontwerpen en implementeren van concurrent programma's

Lex Bijlsma[1)], Kees Huizing[2)], Ruurd Kuiper[2)], Harrie Passier[1)], Harold Pootjes[1)] en Sjaak Smetsers[3)]

1) Faculteit Management, Science and Technology, Open Universiteit , 2) Faculteit Mathematics and Computing Science, Technische Universiteit Eindhoven, 3) Instituut Computing and Information Sciences, Radboud Universiteit

Inleiding

Het ontwerpen en implementeren van een concurrent programma is moeilijk. Het vereist bijvoorbeeld kennis van het executiemodel, inzicht in de onvoorspelbare interactie tussen concurrerende threads en kennis van de primitieven voor synchronisatie en communicatie en de bijbehorende (Java) syntax. Twee voorbeelden van specifieke oorzaken zijn het ontbreken van een geschikte methode om de werking van een concurrent programma weer te geven (docenten gebruiken nu eigen ad-hoc tekenmethoden zonder een duidelijke semantiek) en het ontbreken van een systematiek voor de keuze van een geschikt synchronisatiemechanisme om een raceconditie te voorkomen. Meer algemeen constateren we dat programmeerboeken vooral de syntax behandelen gevolgd door enkele voorbeelden van gebruik. Op *hoe* een (concurrent) probleem moet worden aangepakt wordt niet of nauwelijks ingegaan. Voor studenten blijft dan alleen de trial-and-error methode over.
Een aantal kleine experimenten (enkele hard-op-denken en pair-programming sessies) heeft ons duidelijk gemaakt dat het ontbreken van een methode tot chaotische aanpakken leidt bij studenten en dat het beschikken over een systematische aanpak hier enorm kan helpen.
Om studenten gerichter te laten werken, hebben we een eerste stap gezet in de ontwikkeling van een methode voor het ontwerpen en implementeren van concurrent programma's en hiervoor cursusmateriaal uitgewerkt. Momenteel proberen we dit cursusmateriaal uit in onze colleges en practica. In de volgende paragrafen gaan we kort op deze methode in.

Didactische aanpak

Als didactische aanpak is gekozen voor het model 4C/ID [1]. Hierin wordt onderscheid gemaakt tussen:

- Een taakbeschrijving (wat moet de student doen en wat moet worden bereikt)
- Ondersteunende informatie (de conceptuele kennis die nodig is om het probleem op te lossen en te weten wanneer een oplossing goed is)
- Procedurele informatie (een beschrijving van de (vuist-)regels, stappen, procedures om het probleem op te lossen)
- Oefeningen (van voorbeelden tot geheel zelf een probleem oplossen)

In het vervolg gaan we kort in op de ondersteunende en procedurele informatie.

Ondersteunende informatie

In dit onderdeel wordt eerst uitgelegd wat concurrency is onafhankelijk van een programmeertaal, wanneer concurrency zinvol toegepast kan worden en welke concepten een rol spelen, zoals de call stack, (non) determinisme en atomiciteit. Vervolgens komen Java threads aan de orde en wordt ingegaan op het creëren, starten, laten slapen, joinen en stoppen van threads. Vervolgens wordt duidelijk gemaakt dat door het introduceren van threads nieuwe problemen kunnen ontstaan, namelijk raceconditities en deadlocks. In de huidige versie van ons onderwijsmateriaal gaan we alleen in op het opsporen en voorkomen van raceconditities.
Belangrijk onderdeel van de methode is het gebruik van het UML klasse- en activiteitendiagram tijdens het analyseren en ontwerpen. Het UML activiteitendiagram hebben we voor het opsporen van mogelijke raceconditities uitgebreid: De objecten die door meerdere threads worden gedeeld zijn hierin expliciet opgenomen waarmee het opsporen van lees- en schrijfoperaties op deze objecten grafisch wordt ondersteund.

Procedurele informatie

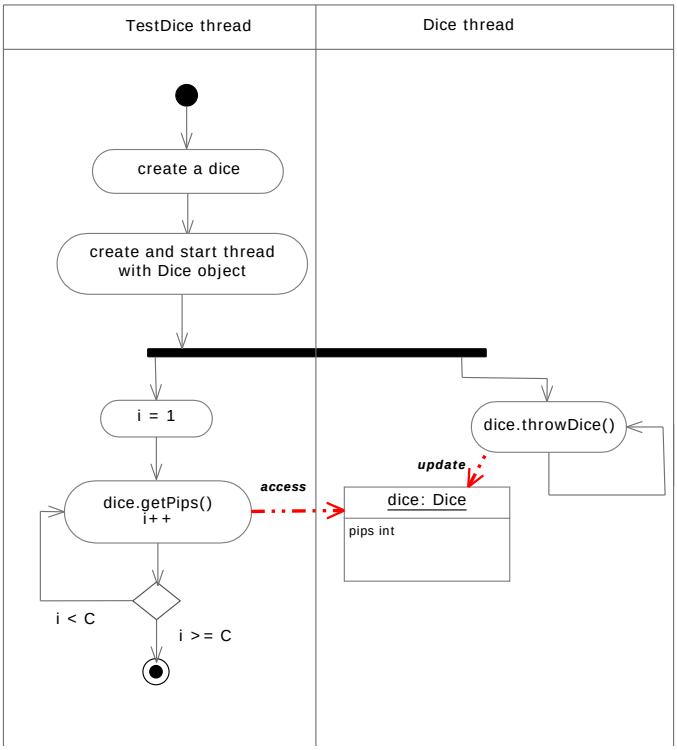
Het belangrijkste ontwerpprincipe in onze aanpak, met een focus op raceconditities, is het toepassen van elementaire synchronisatieprimitieven waarmee de toegang van threads tot de gedeelde objecten

wordt gereguleerd. De voorgestelde ontwerpmethode bestaat uit een stappenplan waarin genoemde UML diagrammen een cruciale rol spelen. We onderscheiden de volgende stappen:

1. Analyse: Is het probleem geschikt om met threads op te lossen? Threads zijn geschikt in situaties van efficiëntie, inherent parallelle processen, responsiviteit en simulatie.
2. Ontwerp: Teken een UML klassendiagram voor de situatie zonder concurrency.
3. Ontwerp: Breid het diagram uit door aan te geven welke klasse interface Runnable implementeert en welke klasse de thread(s) creëert en start.
4. Ontwerp: Teken een hoog niveau activiteitendiagram voor het programmadeel waar methode run actief is. Geef specifiek aan welke objecten worden gedeeld d.m.v. access en update aanroepen.
5. Analyse/ontwerp: Check op racecondities en pas op de juiste manier synchronisatie toe (het Java monitorpatroon).
6. Implementeer het ontwerp.
7. Reflecteer over gekozen oplossing.

Een voorbeeld van een activiteitendiagram

In de aanpak speelt het tekenen en het analyseren van het activiteitendiagram een belangrijke rol. Figuur 1 toont een voorbeeld waarin herhaaldelijk een dobbelsteen wordt gegooid. De linkerkolom bevat de activiteiten van de thread TestDice, de rechterkolom van de Dice thread. In de thread TestDice wordt eerst een dobbelsteen gecreëerd en vervolgens wordt in de Dice thread met deze dobbelsteen herhaaldelijk gegooid. Het dobbelsteenobject met gedeeld attribuut pips is apart opgenomen en duidelijk is te zien dat hier potentieel sprake is van een raceconditie.



Figuur 1. Een uitgebreid UML activiteitendiagram

De gehele methode is beschreven in een technisch rapport [2] dat verkrijgbaar is bij de auteurs.

Literatuur

1. Jeroen J.G. van Merrienboer, Paul A. Kirschner. Ten Steps to Complex Learning, a systematic approach to four-component instructional design. Taylor & Francis, New York, NY, USA, second edition, 2013.
2. Lex Bijlsma, Harrie Passier, Harold Pootjes, Sjaak Smetsers, Didactics for Methodical Concurrency Design, Technical Report (OU, RU, TUE), 2015.