Projekt: Sorting Industrial Robot

Dato: 01-06-2012

Titel:

Systemarkitektur

for

Sorting Industrial Robot

RoboGO

Versionshistorik

Ver.	Dato	Initialer	Beskrivelse
0.1	02-03-12	RHT	Første udkast lavet fra sidste projekt.
0.2	25-05-12	SLT	Tilføjet definitioner og system oversigt.
0.3	30-05-12	SLT, RHT	Tilføjet process view og deployment view.
1.0	01-06-12	ALLE	Endelig version af dokumentet.

Indholdsfortegnelse

Introduktion

Formål

Formålet med dette systemarkitektur-dokument er at dokumentere designet af SIR. De væsentlige aspekter af designet er specificeret heri, og man kan ved at læse dette dokument opnå et overblik over designet.

Referencer

[1] Kravspecifikation

[2] Doxygen generet kode dokumentation.

Definitioner

GUI = Graphical User Interface – også kendt som den grafiske brugergrænseflade.

WPF = Windows Presentation Foundation.

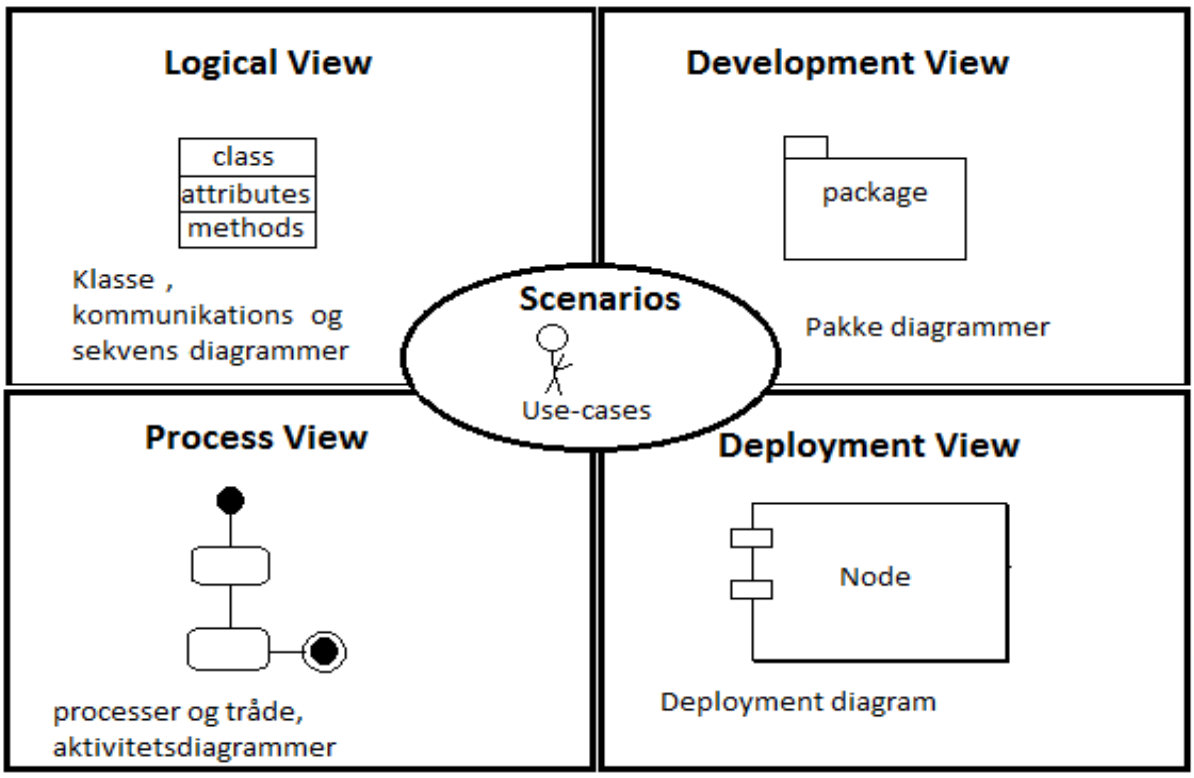
3D = Tre dimensionelt.

SIR = Sorting Industrial Robot

Dokumentstruktur og læsevejledning

Vi har taget udgangspunkt i 4+1 modellen, som illustrerer forskellige måder at vise softwarearkitekturen på.

Illustration 1: 4+1 modellen



Dokumentets rolle i en iterativ udvikling

Selve dokumentet består af dokumentationen fra de fire iterationer.

I iterationerne er der arbejdet med:

- Udarbejdelse af use-cases i kravspecifikation.
- Design af use-cases.
    1. Sekvensdiagrammer.
    2. Klassebeskrivelser (Doxygen).
    3. Klassediagrammer.
- Implementering af use-cases.
- Unit test og integrationstest.
- Accepttest udarbejdet ud fra kravspecifikationen.
- Udarbejdelse af projektrapport.

System oversigt

System kontekst

Se kravspecifikation for aktør-kontekst diagram.

System introduktion

Systemet har det formål, at den skal lade en bruger af systemet styre en robot gennem den grafiske brugergrænseflade. Robotten kan foretage målinger af klodser, enten manuelt, eller brugeren kan skrive et script der automatiserer, alt dette sker igennem en GUI.

Robottens bevægelser kan foretages i et 3D-plan, og her kan robottens hånd åbne og lukke, når den skal tage et objekt. Derudover eksisterer transportbåndet, som også er styrbart fra systemet, og her skal det være muligt at aflæse en indbygget sensor til transportbåndet. Grunden til dette ligger i, at sensoren skal kunne registrere, når der er en klods, der er klar til at blive behandlet af robotten ved vægtmåling vha. vejecellen samt måling af objektets rumfang.

Systemet skal løbende logge systemevents, eftersom det vil være en fordel for brugeren at følge med i, hvad der foretages under systemets kørsel.

Endvidere er det påkrævet af systemet, at det skal være muligt at køre sorteringsprocessen på baggrund af en simulator.

Systemets grænseflader

Grænseflader til person aktører

Grænsefladerne til person aktøren ”Bruger” er lavet ud fra Windows' WPF. Det er gjort muligt, at tastaturet og musen kan anvendes til at interagere med den visuelle brugergrænseflade. Aktøren ”Bruger” kræver et login for at anvende SIR, og når ”Bruger” har logget ind, kan den anvende systemet, dens funktionaliteter og modtage informationer.

Grænseflader til eksterne system aktører

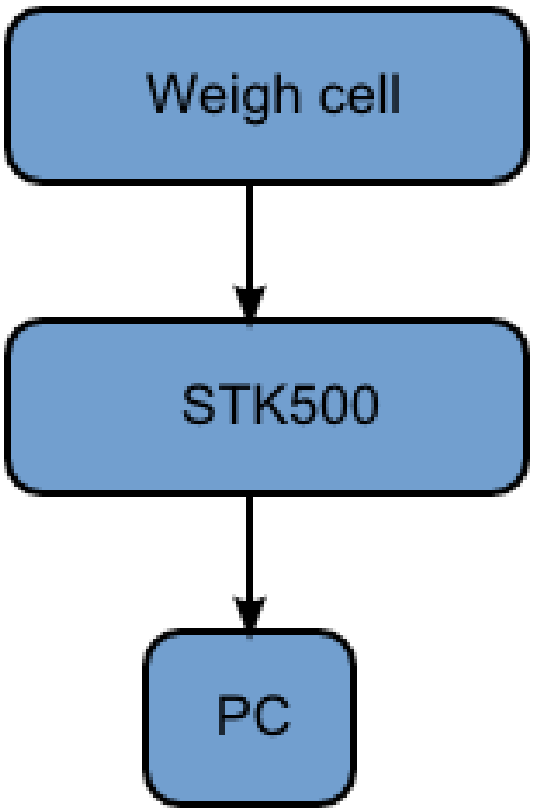


Illustration 2:Her vises kommunikation mellem vejecellen og pc

Grænseflader til hardware aktører

Controller USB, RS232.

Grænseflader til software aktører

USBC.DLL

Use case view

Oversigt

Se use cases under Kravspecifikationen.

Logisk view

Oversigt

I vores implementering af systemet er der så vidt muligt forsøgt at opdele implementering over flere pakker.

- GUI / View
- ViewModel
- SystemControl
- SQLInteraction

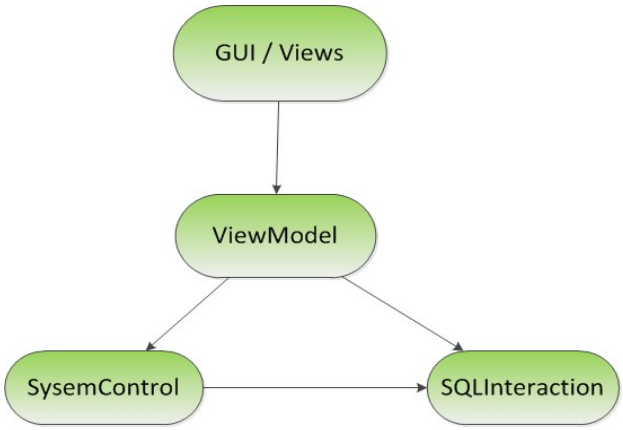


Illustration 3: Oversigt over lagerne i kodesystemet

De forskellige pakker har så vidt muligt kun afhængigheder med ting på samme niveau eller længere nede i hierarkiet, så opdelingen fungerer som lagdeling. Til denne lagdeling bruges der MVVM, dette giver også en separation af vores logik og data præsentation, hvilket også gør det muligt for os at skifte de forskellige lag ud senere hen, hvis der skulle være behøv for det.

GUI:

I denne del er alt det, der skal ses på GUI'en i Views laget. Alle de grafiske funktionaliteter betjenes og ses herfra. Disse indebær manuelt styring, IDE, simulation og database.

ViewModel:

Her blev den logik som View benytter sig af implementeret. Her kan der blandt andet nævnes IDE og Simulator, som indeholder commands hvilket vi bruger til specifikke operationer.

SystemControl:

Her har vi f.eks. robot klassen, denne klasse tager sig af meget generel funktionalitet for projektet, idet vi bruger de wrappede funktioner af .DLL filen, disse kan så bruges til at styre robotten.

SQLInteraction:

Denne del tager sig at at oprette forbindelse til databasen og hente værdier derfra.

Klasse diagram

Systemets klassediagram blev autogenereret af Visual Studio 2010's Architecture funktionen. Diagrammet viser opbygningen af systemet og klassernes individuelle afhængigheder. Diagrammet fylder meget, derfor henvises til bilag.

Sekvens diagrammer

Use Case 3: Foretage manuel styring

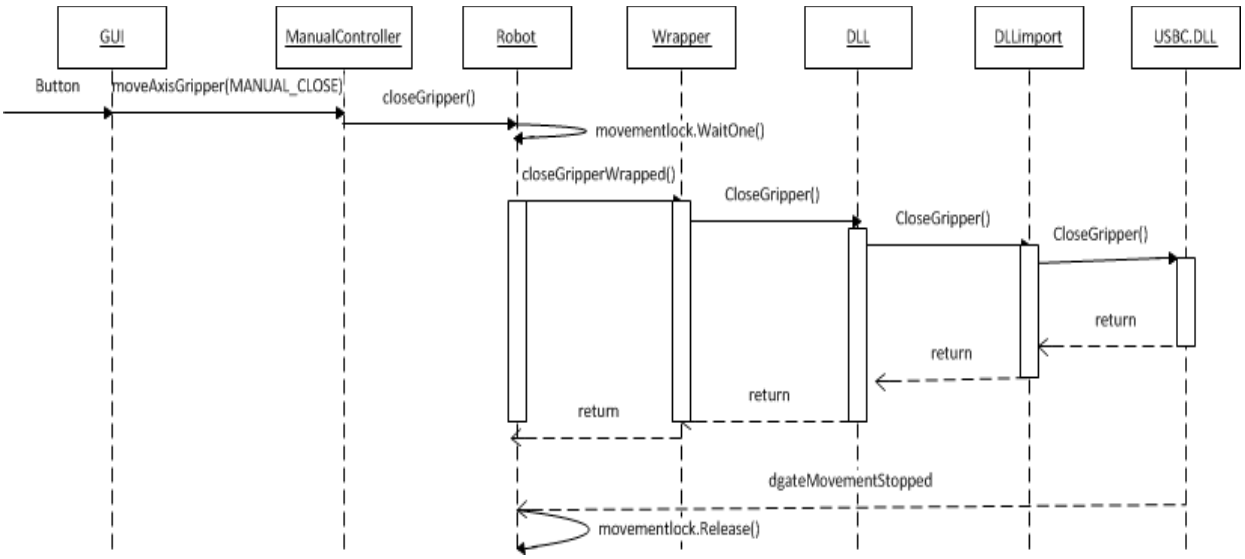


Illustration 4:Sekvensdiagram for usecase 3

Beskrivelse:

Hvis brugeren ikke ønsker at bevæge robotten rundt, men derimod bare ønsker at åbne gripperen, ses det her. Brugeren trykker på Close Gripper, som så bliver sendt videre. Vi ser i Robot, at movementlock.WaitOne() bliver kaldt, dette er vores semaphore, som vi bruger i forbindelse med WatchMotion, dette sørger for synkronisering, dvs. der er ingen andre funktioner som kan gå ind og begynde at blive eksekveret før at vi har fået et callback. Udover synkroniseringsmekanismen med WatchMotion, ser vi hvordan closGripper bliver kaldt igennem de forskellige lag. Vi har Wrapper, hvori funktioneren fra DLL filen er blevet wrappet til C# kode, derfter går vi videre ned i lagene, til vi til sidst når USBC.DLL, så sker det førnævnte callback. Til at modtage dette callback, bruges der en delegate oppe på Robot niveau, når denne modtages, slipper vi den semaphore vi tidligere tog.

Use Case 12: Måle vægten af objekter.

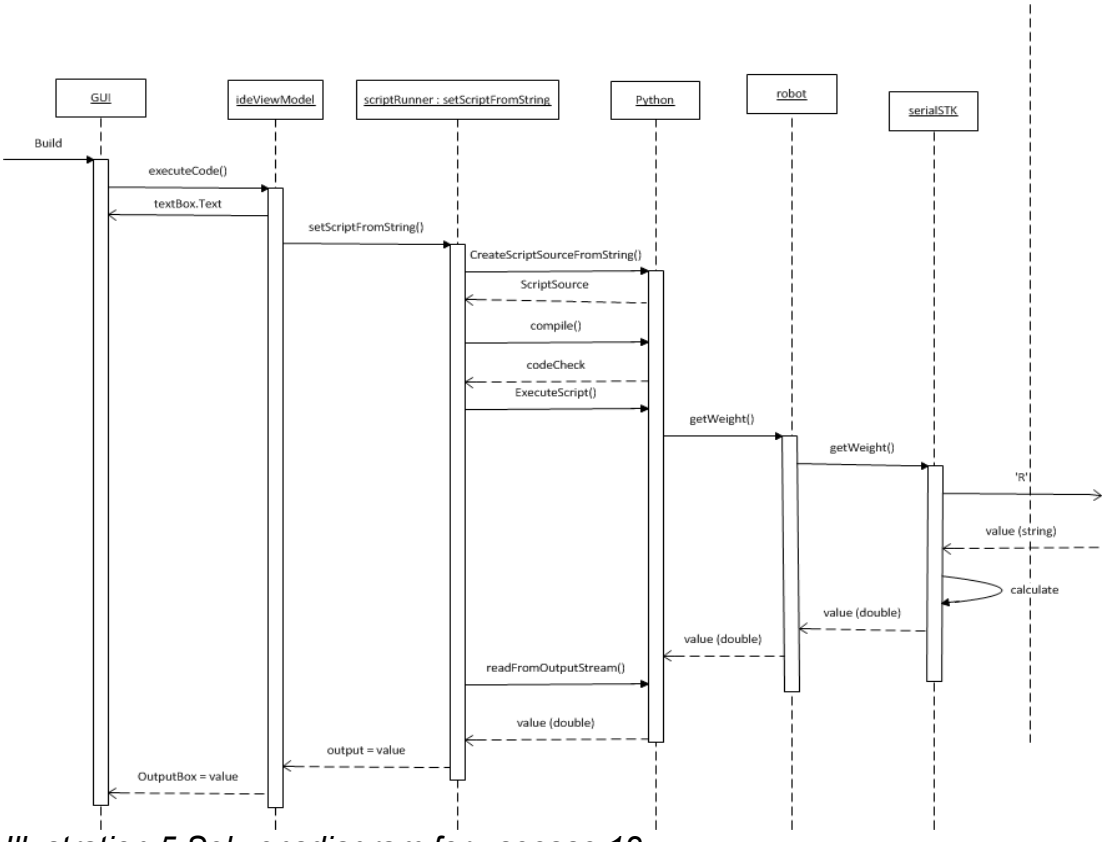


Illustration 5:Sekvensdiagram for usecase 12

Beskrivelse:

Her ses et build kald, hvor der vil hentes vægten af en klods. Brugeren sender en

anmodning via GUI'en som kalder sin binding til IdeViewModel. I ideViewModel kaldes scriptrunner og verificerer kommandoen via python. Derefter kalder den funktionen getWeight() i robot klassen, som kalder readADC i serielSTK. I serielSTK tjekkes COM porten, åbnes og sender et R streng til STKKittet.

STKKittet modtager R strengen og udfører et AD konvertering, hvor der bliver sendt værdien tilbage i COM porten. Værdien bliver konverteret i seriekSTK og returneret.

Process view

Oversigt over processer

I dette view vises, hvorledes de enkelte processer i systemet kommunikerer. Vi har fire tråde, og herunder ses, hvordan de kommunikerer med hinanden.

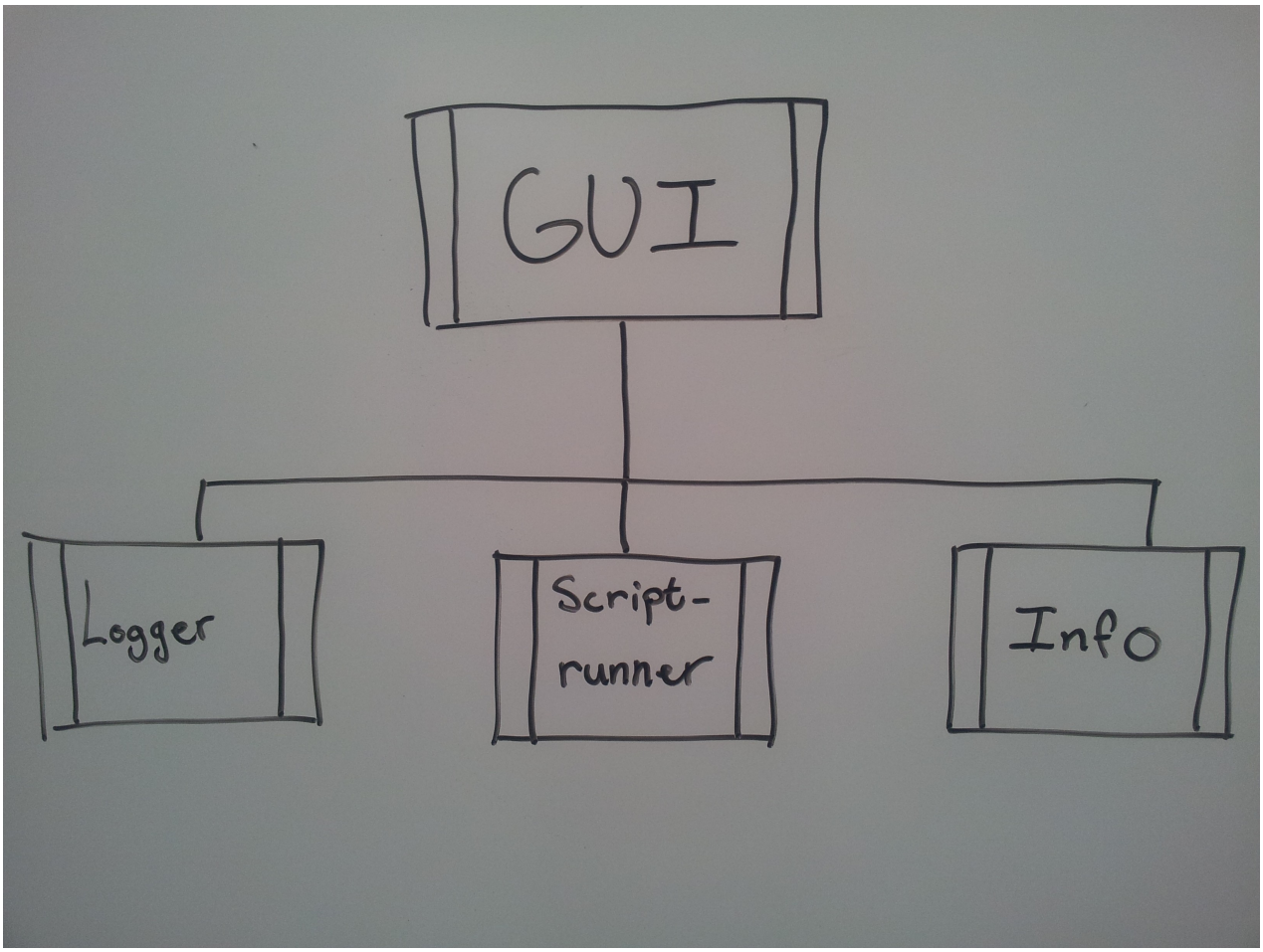


Illustration 6:Process view, hånd tegnet

Implementering

Vi har lavet en tråd klasse(ThreadHandling), hvorfra man laver og kontrollerer trådene i systemet. Dette blev implementeret for at gøre det nemmere at lave tråde samt lukke dem alle ned, når programmet afsluttes, så der ikke var løse tråde til sidst, der kørte i baggrunden.

Selve trådene er så lavet med standard C# tråde(System.Threading.Thread).

Kommunikation og synkronisering

Til at styre trådene har vi den almindelige join funktion i trådklassen.

Til synkronisering af loggeren har vi brugt WaitHandle klassen for at kunne signalere, når der er events, der skal logges.

Ved de klasser, der bliver delt mellem trådene gennem fabrikklassen (Factory), har vi brugt volatile objekter og låst (Lock{}) de områder, hvor man bruger deres instansreferencer. Dette er gjort for at instansreferencerne ikke bliver sat flere gange f.eks. to robot forbindelser bliver lavet.

Procesbeskrivelser

GUI tråd:

Beskrivelse:

GUI tråden startes op, når systemet initieres, og det er også vores main tråd, der starter de andre tråde. Procesfiguren for GUI tråden kan ses i procesfiguren for ScriptRunner.

Levetid:

Fra program start til slut.

Aktiv:

Aktiv hele tiden.

ScriptRunner tråd:

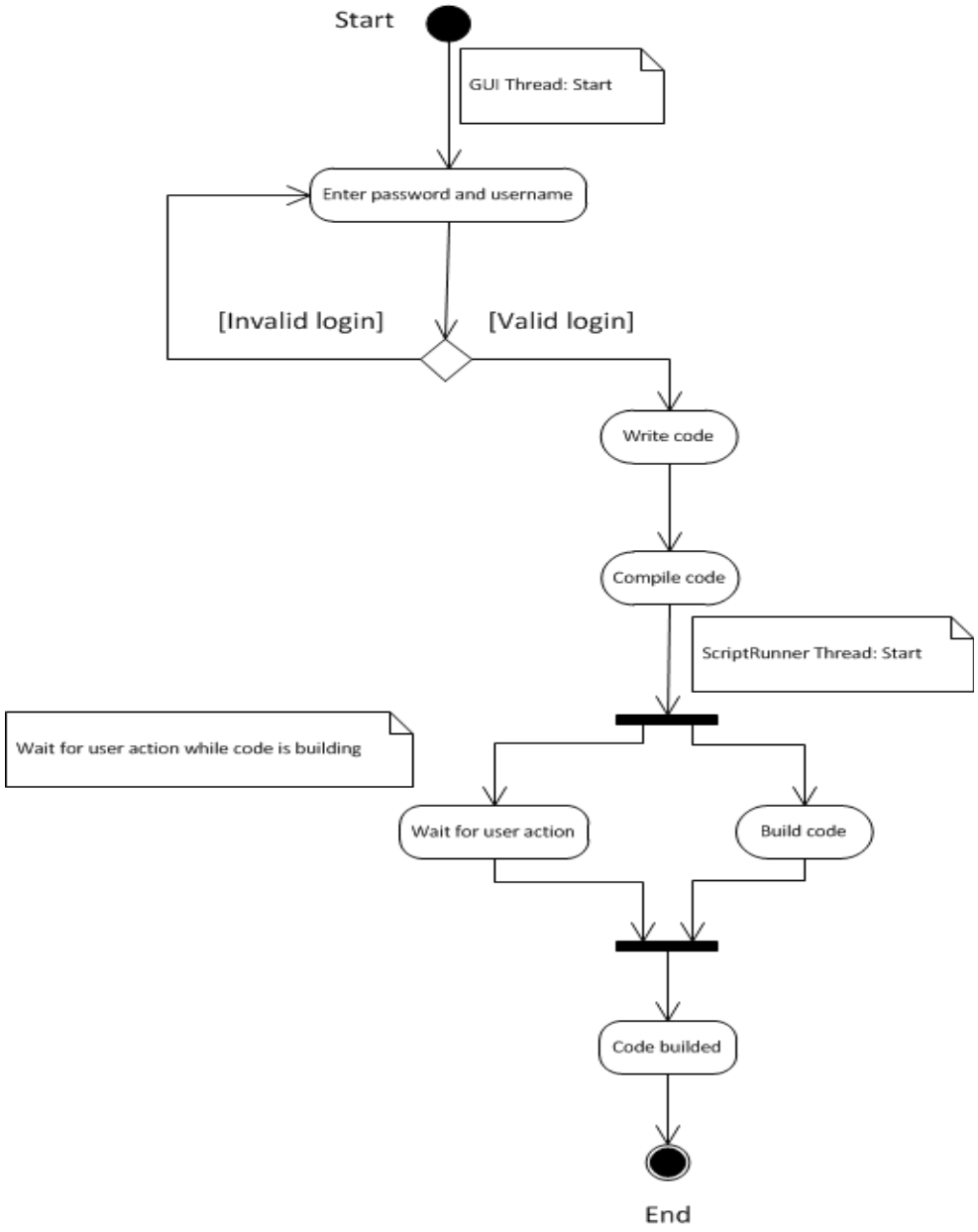


Illustration 7:Aktivitestdiagram for ScriptRunner tråd samt GUI tråd.

Beskrivelse:

Bliver kørt hver gang noget kode fra IDE´en skal udføres.

Levetid:

Fra building til slut.

Aktiv:

Mens koden bliver kørt.

Logger tråd:

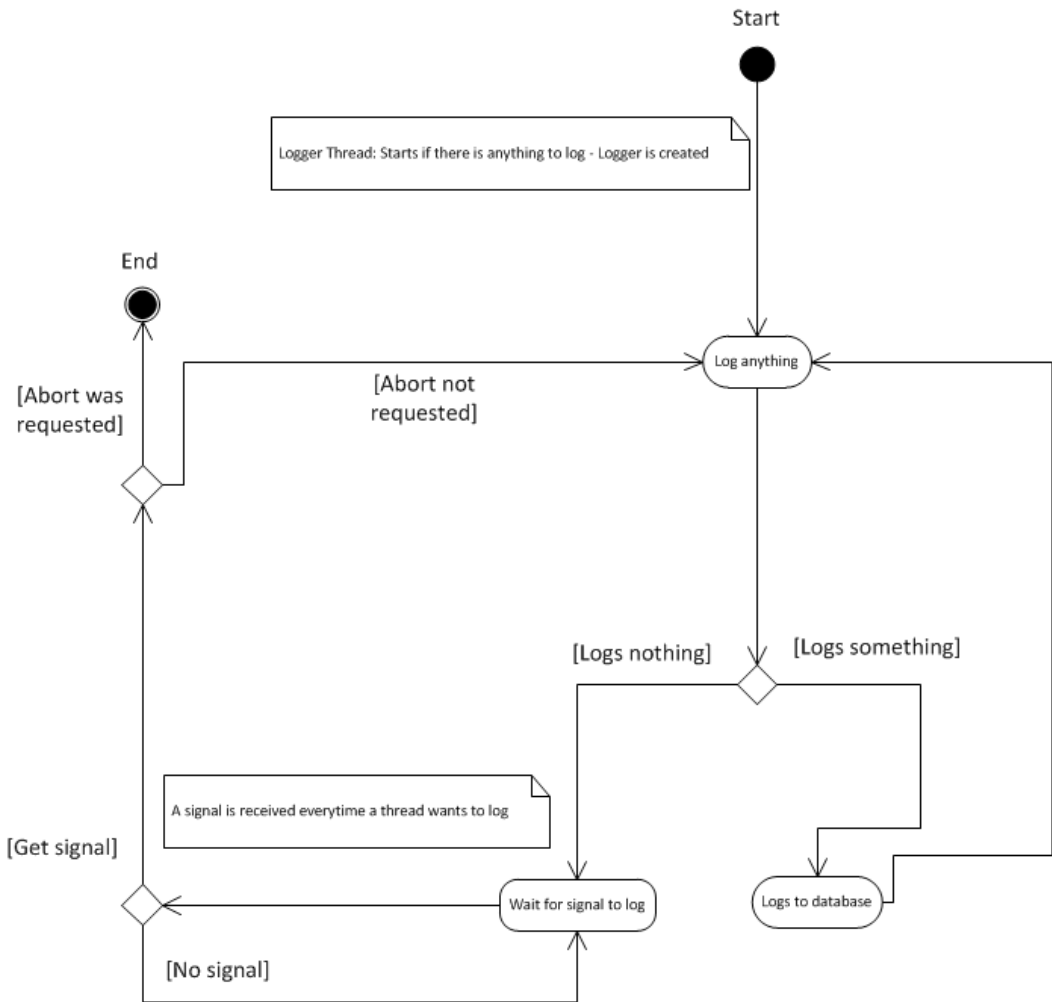


Illustration 8:Aktivitetsdiagram for Logger tråd

Beskrivelse:

Denne tråd bliver startet, når loggeren bliver initialiseret og venter, indtil den bliver bedt om at logge. Logningen sker ved, at der bliver kaldt en funktion på den, som adder en log besked til en liste. Listen bliver så tjekket i log tråden, hvor de én efter én bliver logget ned.

Levetid:

Fra første gang der logges noget.

Aktiv:

Når der er log beskeder i kø, ellers venter den.

Info tråd:

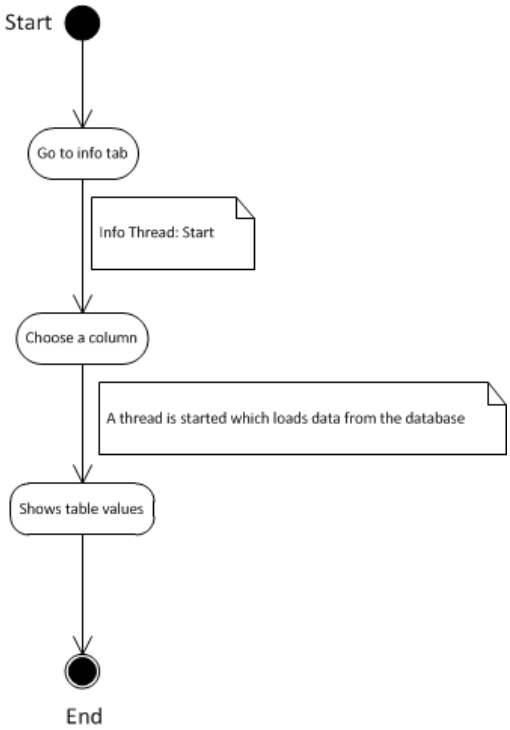


Illustration 9:Aktivitetsdiagram for Info tråd

Beskrivelse:

Info tråden bliver brugt hver gang, der skal hentes informationer ned om tabellerne, der vises i GUI´en. Dette er for, at resten GUI´en ikke fryser fast, mens informationen hentes ned, eller hvis der ikke er database adgang. Det er teknisk set to forskellige tråde, men de minder meget om hinanden – den eneste forskel er, at den ene henter eksisterende tabeller, og den anden henter selve informationen om tabellerne ned.

Levetid:

Når brugeren beder om info fra databasen, til det er hentet ned.

Aktiv:

Mens data hentes ned.

Deployment view

Oversigt over systemkonfigureringer

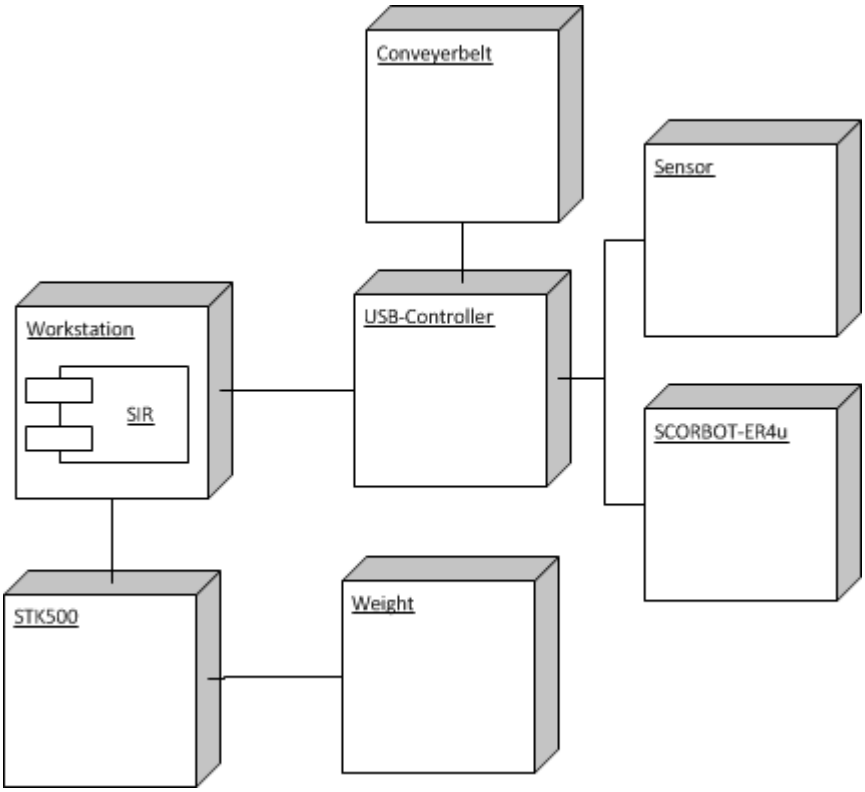


Illustration 10:Deployment diagram

Node beskrivelser

SCORBOT-ER4u:

Robotten som udfører operationerne til sorteringsprocessen.

USB-Controller:

Forbindelse mellem 'workstation' og andre komponenterne som kan styres fra programmet eller hentes værdier fra.

STK500:

Mellemled mellem 'workstation' og 'weight', så der kan tages målinger fra programmet.

Workstation:

Computeren hvor programmet(SIR) køres på.

Sensor:

Lys sensor, som registrerer at der er et objekt eller ej foran den.

Conveyerbelt:

Transportbånd til transport af klodserne, som skal sorteres. På den er monteret 'sensor'.

Weight:

Vægt som der kan tages målinger.

Development view

Oversigt

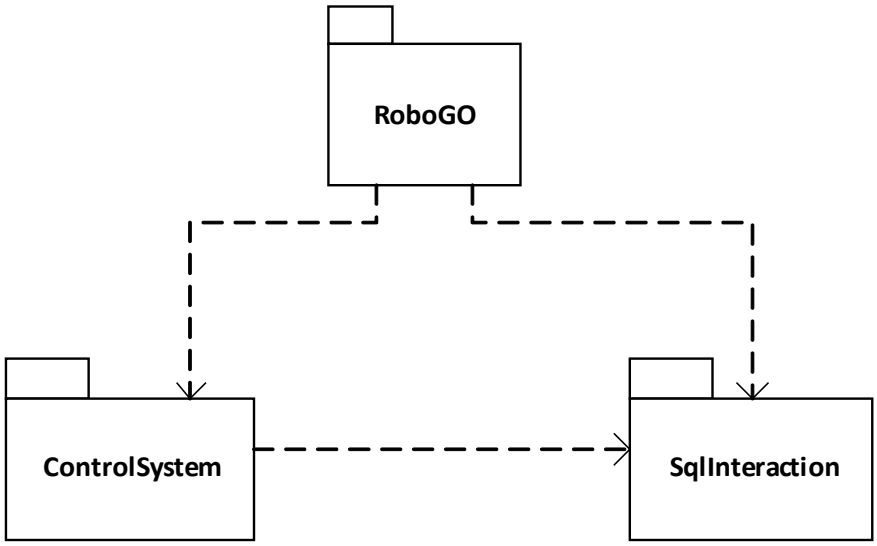


Illustration 11:Package diagram

RoboGO package indeholder GUI og viewmodel klasser.

ControlSystem indeholder alle de funktionelle klasser.

SqlInteraction indeholder klasser, som kommunikerer med databasen

Komponentbeskrivelser

Se dokumentation på CD´en for kode dokumentation("kode autogen dokumentation.pdf").

Generelle arkitekturbeslutninger

Dette afsnit beskriver de beslutninger vi har taget om arkitekturdesign.

Arkitekturmål og begrænsninger

Som krav og begrænsninger er der blevet sat følgende, der skal overholdes:

- En database skal kunne opbevare og manipulere relevante data fra processen.
- En IDE skal kunne fremstille programmer til styring af robottens proces.
- En simulator skal kunne simulere den fysiske robot, således at der kan skiftes mellem den 'ægte' robot og simulatoren.
- Et styreprogram udformet som GUI, der programmeres i C#.
- Et system, der kan måle klodsers masse og rumfang.

Arkitektur mønstre

Vi har valgt at bruge MVVM pattern til at implementere programmet. Dette vil gøre arbejdet lettere for designeren, som ikke skal beskæftige sig med codebehind. Designeren skal kun binde til viewmodel. Ved at flytte funktionerne til viewmodel, ville de være mere testbar.

Dette er blevet valgt da C# med WPF giver relativ nem mulighed for at implementere med MVVM pattern. Ting som DataBinding og Commands.

Andre patterns der er blevet brugt er Singleton og Factory sammen med Indirection for at fjerne afhængighed fra forskellige komponenter i systemet samt at flere klasser havde brug for at deles om de samme klasse instanser.

Generelle regler for brugergrænseflade

Systemet skal være intuitivt, og gennem brugergrænsefladen skal det kunne være muligt for den pågældende bruger at skabe et program, som kan gemmes. Udover at gemme filer skal man kunne åbne en eller flere filer, og vi skal også have mulighed for at kunne skifte mellem de to instanser for hhv. robot og simulator. Endvidere skal der være mulighed for at styre robotten manuelt gennem GUI'en, og sidst skal der også være mulighed for brugeren at manipulere med de data, der er persisteret i databasen.

Fejlhåndtering

Vi har implementeret fejlhåndtering ved , at systemet gennem GUI'en informerer, hvis der er lavet en ustabil opstart. Hvis de nødvendige filer m.m. ikke er initialiseret ordentligt, kan dette dog tjekkes ved at tjekke forbindelsen mellem GUI'en og robotten.

Implementeringssprog og værktøjer

Implementeringssprog har primært været C#, men der har også været anvendt SQL og C.

Til programmet og dokumentation:

- Visual Studio 2010 hvor der er brugt C# og IronPython
- GIT til versionskontrol
- LibreOffice til dokumentation
- Visio til UML tegninger
- InkScape

Til databasen:

- Microsoft SQL Server Management Studio
- Database Design Studio Lite (DDS Lite)

Andet:

- CodeVision for STK500 programmering
- STK500 for målinger fra vejecellen

Implementeringsbiblioteker

- USBC.dll
- NUnit.
- DotCover.
- MVVMToolkit

Størrelse og ydelse

Programmet er udviklet til at køre på Windows platformen, på en standard computer med .NET 4.0 installeret.

Oversættelse

Oversættelse-hardware

Computer med minimum XP(Service Pack 2)(32/64-bit) og installeret SDK for .NET 4.

Herudover skal bruges STK500 på grund af vejecelle kommunikation.

Oversættelse-software

Selve I4PRJ4 Robot programmet kan afvikles på en Windows maskine med .NET installeret.

Windows maskinen skal være på skolens netværk for at kunne etablere forbindelse til skolens database server, og dermed forbinde til systemets database.

For C-programmeringsdelen skal alle kildefiler lægges i samme mappe. I Codevision laves et nyt projekt, som gemmes i samme mappe, og kildefilerne tilføjes projektet. Projektet konfigureres, så det er sat op til ATmega16-chippen og en chipfrekvens på 3,6864 MHz. Derefter kan koden kompileres i Codevision.

Installation

Programmet køres ved at eksekvere RoboGO.exe i mappen med alle de nødvendige filer, som støtter op til RoboGO-projektet.

Kørsel

Kørsels-hardware

SCORBOT-ER4u med tilhørende USB-Controller, vægt og vejecelle, som er blevet udleveret til os.

Kørsels-software

Vores SIR program med USBC.dll fra SCORBASE, som vores interface til USB-Controlleren.

Start og stop

Sker ved manuel opstart af programmet og afslutning af dette.

Brugergrænseflade

Brugergrænsefladen lader brugeren logge ind, når programmet startes. Herefter kan brugeren efter gyldig login vælge, om der skal køres en robot eller simulator. Brugergrænsefladen vil endvidere være den del af systemet, som brugeren vil interagere med, da styringen af robotten, skrivning af kode, simulationen og logging vil være at finde i brugergrænsefladen.

Bilag

Tjek medhørende CD for at se eventuelle bilag.