AARHUS UNIVERSITET

Speciale

Udvikling af et framework til dynamiske interaktive historier

- Digital historiefortælling med mobiltelefoner og Arduino-enheder

Forår 2010

Teknisk it (Distribuerede Realtidssystemer)

Nicolaj Bjerregaard Christensen (20013595)

Afleveret d. 4. juni 2010

Nicolaj Bjerregaard Christensen,
Studerende,
20013595

Frank Allan Hansen,
Vejleder

Creating a framework for dynamic interactive stories

- digital storytelling with mobile phones and Arduino-based systems

Abstract

In this thesis a framework for dynamically expanding interactive stories will be created. The framework will be created partly as a configuration protocol for graphical user interfaces and partly as a communication protocol for creating interaction between mobile and embedded systems.

Interactive stories give the user (or reader) the ability to affect the storyline and helps communicate information through a greater immersion into the story. The interaction in these stories is normally predefined and the user's perception of being able to alter the story is more often an effect of the storytelling and the plot than it is real interaction. A long line of research projects has been working on creating frameworks to enable easier and better production of these interactive stories.

The ability to interaction with the environment through the story, such as a story which takes place in the same urban setting as the user is in, is another element often incorporated into interactive stories. In mobile stories is can be difficult to incorporate physical interaction and the interaction often ends up as being location based or complete decoupled from the storytelling (in cases where the user has to "exit the story" and manually interact with the surrounding environment). In these cases the real-world interaction is not done through the storytelling media.

The background for this thesis is a desire to explore the possibility of supporting both dynamically expanding stories as well as interaction with the physical surroundings directly from the story. The story can be expanded or altered by the user's movement through, or interaction with, the surrounding environment and physical interaction can be implemented by interacting with embedded units placed in the environment.

The study contains an analysis of related work in the area and an analysis of an existing story framework. The result of the analyses is implemented in two separate protocols, which makes it possible to expand an ongoing story and interact with the surrounding environment. The protocols are implemented both as expansions to the existing framework and as a new framework for the embedded Arduino units.

Finally a proof-of-concept prototype is implemented that is used to verify the protocol designs as well as demonstrate the ability to dynamically expand an interactive story. The prototype consists of both a mobile interactive story and interaction with embedded Arduino based units. In this prototype examples of both story expansion, reading of sensor values from an embedded system and interaction with an electrical mini aquarium (with the ability to control both light and movement in the aquarium) are given. All interaction is handled directly from within the story as it is expanded based on the user's physical movements.

The result of the thesis is a software framework which makes it possible to create dynamic interactive stories based on interaction with embedded units.

Resume

I dette speciale bliver der udviklet et framework til dynamisk udvidelse af interaktive historier, der dels udvikles som en konfigurationsprotokol til grafiske brugergrænseflader og dels som en kommunikationsprotokol til at skabe interaktion mellem mobile og indlejrede enheder.

Interaktive historier giver brugerne mulighed for at påvirke historiens forløb og er med til at formidle information på bedre måde gennem en større indlevelse i historien. Almindeligvis er interaktionen i disse historier defineret på forhånd og brugerens oplevelse af at kunne påvirke historien er ofte en effekt af historieformidlingen og selve plottet, mere end det er en reel interaktion. En lang række forskningsprojekter har arbejdet med at udvikle frameworks som kan gøre udviklingen af disse historier nemmere og bedre.

Mulighed for at lave interaktion med miljøet igennem historien, f.eks. i en historie der foregår i et det urbane miljø som brugeren bevæger sig i, er også et emne der ofte bliver inkorporeret i interaktive historier. Fysisk interaktion er problematisk at indbygge i mobile historier og det ender ofte med at interaktionen er lokationsbestemt eller afkoblet fra historieformidlingen. F.eks. ved at brugeren skal bevæge sig ud af fortællingen og interagere med den ”almindelige” verden, uden at denne interaktion udføres igennem det medie som historien formidles med.

Baggrunden for specialet er et ønske om at undersøge muligheden for at understøtte både dynamisk udvidelse af interaktive historier (ved at historien udvides eller forandres baseret på brugerens bevægelser og opførsel i det fysiske miljø) og interaktion med det fysiske miljø direkte fra historien (ved at interagere med indlejrede enheder der er placeret i miljøet).

Undersøgelsen omhandler en analyse af relateret arbejde indenfor området, samt en analyse af et eksisterende historieframework. Resultatet af analyserne implementeres i to protokoller, der giver mulighed for at udvide en igangværende historie og interagere med indlejrede enheder. Protokollerne implementeres både som udvidelser til det eksisterende framework til Java ME og som et nyt framework til indlejrede Arduino-enheder.

Endeligt implementeres der en proof-of-concept prototype, som benyttes til at verificere protokollernes resultater samt at demonstrere mulighederne for at udvide interaktive historier. Prototypen benytter sig både af en mobil interaktiv historie og interaktion med en række indlejrede Arduino-enheder. I prototypen vises der både eksempler på udvidelse af en historie, aflæsning af sensorværdier fra en indlejret enhed samt interaktion med et elektrisk miniakvarium hvor det er muligt at tænde og slukke for motor og lys direkte fra historien.

Resultatet af specialet er et softwareframework som gør det muligt at lave dynamisk udvidelse af interaktive historier, baseret på interaktion med indlejrede enheder.

Indholdsfortegnelse

Kapitel 1. Indledning

Dette kapitel indeholder en generel indledning til specialet, baggrunden for det, samt en beskrivelse af hvad formålet med det er. Derudover indeholder afsnittet også en afgrænsning af specialet og en beskrivelse af ideen bag implementeringen. Til sidst beskrives indholdet i projektet; hvad det består af, hvordan det udføres og hvilket resultat der forventes. Derudover dokumenteres hvilken fremgangsmåde der anvendes for at nå målet.

1.1 Baggrund

1.1.1 Interaktive historier

Interaktive historier er et emne der bliver forsket meget i. Ideen bag de interaktive historier er ofte enten at lave en historie hvor "læseren" er en del af den fortalte historie eller hvor "læserens" valg har betydning for udfaldet/forløbet af historien, at læseren dermed kan interagere med historien. Nogle forskningsprojekter har også begge elementer med i historien (Riedl, Stern 2006, Gustafsson et al. 2006, Hansen, Kortbek & Grønbæk 2008).

De interaktive elementer skal normalt være defineret på forhånd og systemerne er ikke beregnede til at blive omkonfigureret efter de er blevet startet. Brugeren kan derfor opleve historien som værende en fortælling mere end en interaktiv historie.
Forsøg på at kombinere forskellige fysiske interaktionsmuligheder og lave en "historie", der automatisk udvides efterhånden som brugeren kommer frem, ender ofte med at blive interaktive spil i stedet for en interaktiv historie (Gustafsson et al. 2006, Scheible, Tuulos & Ojala 2007, Bichard et al. 2006).

Mulighed for løbende at omkonfigurere en historie vil give flere muligheder for interaktion og samtidig gøre det muligt at lave interaktive historier der ikke er veldefinerede på forhånd eller som består af en række ukorrelerede elementer, f.eks. en rundvisning på et museum.

1.1.2 Fysisk kobling mellem historie og mobil

Den fysiske kobling mellem læseren og historien implementeres ofte igennem mobiltelefoner, da det er en velkendt platform der har stor udbredelse og giver mange muligheder. Interaktionen med historien benytter så de forskellige muligheder det mobile medie stiller til rådighed; f.eks. kamera, GPS, afspilning af lyd, SMS m.m. Igennem disse interaktionsmuligheder kan brugeren så interagere med historien og opleve den som interaktiv (Flintham et al. 2007, Reitmaier, Marsden 2009).

Paay et al. har f.eks. benyttet lokationsbestemmelse (via GPS-enheder i en PDA) til at opnå fysisk interaktion og dynamisk at opbygge en historie baseret på hvor brugeren bevæger sig hen (Paay et al. 2008).

Et problem med fysisk interaktion med miljøet, i interaktive fortællinger, er at det ofte kræver at brugerne skal benytte forskelligt udstyr til at interagere med forskellige installationer. Dette medfører problemer som en stejlere indlæringskurve til systemet og flere steder hvor systemet kan fejle (Kurdyukova, André & Leichtenstern 2009).

Ved at samle historien og interaktionen med miljøet på en enkelt mobil platform, opnår man samtidig den fordel at man minimerer mængden af udstyr der skal bruges, samtidig med at man øger sandsynligheden for at brugerne af systemet er vante til at bruge udstyret på forhånd.

Der er også en række projekter der har arbejdet med at udvikle en protokol til automatisk generering af brugergrænseflader. Formålet med disse protokoller var at visualisere eksterne enheders funktionalitet og muliggøre en interaktion med disse enheder, uden at enhederne på forhånd var kendte (Nichols et al. 2002, Gajos, Weld 2004).

1.1.3 Interaktion med indlejrede enheder

Trådløs interaktion med indlejrede enheder kan gøre det muligt at udvide historien intelligent, da disse enheder kan indeholde en stor mængde logik eller endda kommunikere med en fælles server, før de sender næste udvidelse til historien.
En anden fordel, der kan opnås gennem interaktion med indlejrede enheder, er at det bliver muligt at interagere med den fysiske verden på en nem måde, direkte fra historien. Gennem et veldefineret interface kan historien kommunikere med enheden og påvirke den fysiske verden, f.eks. ved at enheden er tilkoblet forskellige fysiske sensorer/aktuatorer.

Derudover giver en sådan platform mulighed for at man kan benytte historiefortællingen på en ny og spændende måde, f.eks. ved at man laver en historie der overhovedet ikke er defineret på forhånd og som afhænger fuldstændig af brugerens valg og færden.

Gennem brugerens færden i det fysiske miljø, vil hun komme forbi forskellige indlejrede enheder, der hver vil kunne opdatere historien med de muligheder og informationer som er tilknyttet den enkelte enhed. Det kan være narrativ, tekst, billeder, fysiske interaktioner, ekstern kommunikation og meget mere. Ved at lade applikationen konfigurere sig løbende, oplever brugeren at hele fortællingen er interaktiv. Dette vil f.eks. kunne benyttes i rundvisninger, orienteringsløb, gruppeopgaver eller lignende, hvor det ikke er en kronologisk fortælling der er hovedformålet.

1.1.4 Interaktiv formidling af information

Hvis man ønsker at benytte multimedier og fysisk interaktion til at formidle information, f.eks. på et museum eller et bibliotek, stiller dette store krav til udformningen, brugen og intuitiviteten af det valgte medie eller den valgte fysiske installation.

Dette emne er blevet undersøgt i en række forskellige projekter, der har til fælles at den fysiske interaktion enten var integreret i udstillingerne (Kortbek, Grønbæk 2008) eller at den krævede at gæsterne benyttede sig af nye interaktionsmetoder i forhold til dem de var vant til (Hornecker, Stifter 2006, Danks et al. 2007).

Ved at benytte sig af de teknikker, der er nævnt i de forrige afsnit, vil det være muligt at integrere den samme form for interaktion på en velkendt mobil platform. Dette kan anvendes når gæsterne medbringer deres egen mobiltelefon og igennem den opnår mulighed for at interagere fysisk med de installationer der er på f.eks. museet.

1.2 Formål

Hovedformålet med specialet er at afklare de problemstillinger der er blevet opstillet i forrige afsnit og udvikle et softwareframework der løser dem.

Muligheden for at benytte mobiltelefoner og indlejrede enheder, til at skabe interaktion mellem historien og det fysiske miljø, skal undersøges.

Ligeledes skal muligheden for at benytte indlejrede enheder til at udvide en igangværende interaktiv historie undersøges.

Formålet med specialet er derfor at undersøge følgende:

Hvordan understøttes interaktive historier med dynamisk udvidelse, igennem fysisk interaktion med indlejrede enheder?

For at afklare hovedspørgsmålet, besvares følgende delspørgsmål igennem projektet:

Delspørgsmål I.	Hvordan repræsenteres indlejrede sensorer og aktuatorer grafisk på en mobiltelefon?
Delspørgsmål II.	Hvordan skabes denne grafiske repræsentation dynamisk, uden at de indlejrede enheder og mobiltelefonen kender hinanden på forhånd?
Delspørgsmål III.	Hvordan håndteres fysisk interaktion med indlejrede enheder fra en mobiltelefon?
Delspørgsmål IV.	Hvordan skabes den fysiske interaktion igennem et interface til indlejrede enheder?
Delspørgsmål V.	Hvordan udvides prototypeimplementeringen med andre former for interaktion og dynamisk historiefortælling?

1.3 Afgrænsning

Projektet afgrænses til kun at omhandle implementeringen af den fysiske interaktion mellem historien/brugeren og de indlejrede enheder. Implementeringen af historien, de interaktive elementer og andet, der kun har med historieformidlingen at gøre, er ikke en del af dette projekt.

De indlejrede enheder, der benyttes i projektet, implementeres på Arduino-platformen[1]. Det er en open source platform, som bruges til at implementere interaktive objekter. Det er en udbredt platform, der bruges til at lave prototyper på større sensorsystemer/interaktive systemer. Platformen har understøttelse for et væld af forskellige sensorer og kommunikationsinterfaces. Derfor afgrænses frameworket til kun at benytte Arduino-platformen.

Den trådløse kommunikation mellem mobiltelefonen og enhederne, bliver implementeret via Bluetooth, da der allerede er eksisterende understøttelse for dette på Arduino-platformen. Bluetooth er samtidig implementeret på stort set alle moderne mobiltelefoner. Der ses derfor bort fra andre trådløse kommunikationsprotokoller i denne prototype.

Den mobile del af frameworket, hvorpå den grafiske brugergrænseflade skal være og selve brugerinteraktionen skal foregå, implementeres på en velkendt og udbredt platform. Da langt de fleste moderne mobiltelefoner understøtter Java ME, der giver mulighed for at implementere en brugergrænseflade som brugerne er velkendte med, afgrænses projektet til kun at blive implementeret på denne platform.

Dette projekt benytter sig af *MOURDA*(Mobile Urban Drama)-platformen (Hansen, Kortbek & Grønbæk 2008), der er blevet udviklet på Aarhus Universitet.
Platformen benytter XML-baserede historiebeskrivelser, der med en vis tilpasning, muliggør at man dynamisk kan indsætte nye historieelementer mens en historie er i gang. Den kan dog ikke på nuværende tidspunkt håndtere disse dynamiske elementer, historier uden et reelt startelement eller hvor rækkefølgen ikke er kendt på forhånd, som beskrevet i baggrundsafsnittet. Mulighed for dette vil blive undersøgt og implementeret i specialet.

Platformen giver samtidig en mulighed for at man, gennem en XML-beskrivelse, kan definere hvorledes en brugergrænseflade skal se ud, samt en mulighed for at definere hvilke interaktioner der er og deres resulterende handling.

[1] http://www.arduino.cc/

1.4 Frameworkbeskrivelse

Dette speciale undersøger mobil fysisk interaktion med indlejrede enheder, samt koblingen mellem denne og mobile interaktive historier. Implementeringen af dette håndteres igennem et softwareframework, der beskrives herunder.

1.4.1 Krav til frameworket

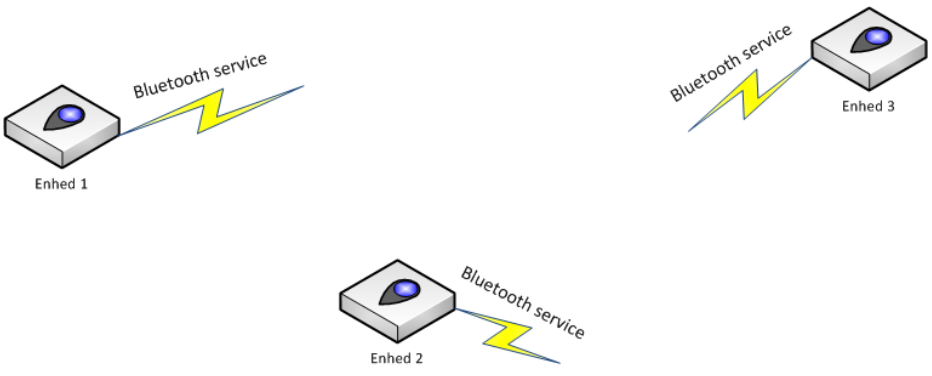
Frameworket skal gøre det nemt at udvikle systemer, der ligger inden for emnet beskrevet i afsnit 1.1 *Baggrund*, ved at opfylde følgende krav:

- Det skal være muligt at starte en historie der ikke er komplet og som på forhånd ikke indeholder alle nødvendige elementer.
- Historier skal kunne udvides dynamisk ved at interagere med indlejrede enheder i miljøet.
- Det skal ligeledes være muligt at starte en ny historie ved at interagere med enheder.
- Frameworket skal gøre det muligt at forbinde til sensorer/ aktuatorer, der er forbundet til indlejrede enheder, og igennem interfacet påvirke disse.
- Det skal være muligt for en ekstern enhed at konfigurere brugergrænsefladen, i forhold til de muligheder den enkelte enhed stiller til rådighed.
- Opdagelsen af, og oprettelsen af forbindelse til, en ekstern enhed, skal foregå transparent for brugeren. Frameworket skal gøre det muligt at lede efter, finde og forbinde til enheder løbende, mens en applikation er aktiv.

1.4.2 Baggrund for frameworket

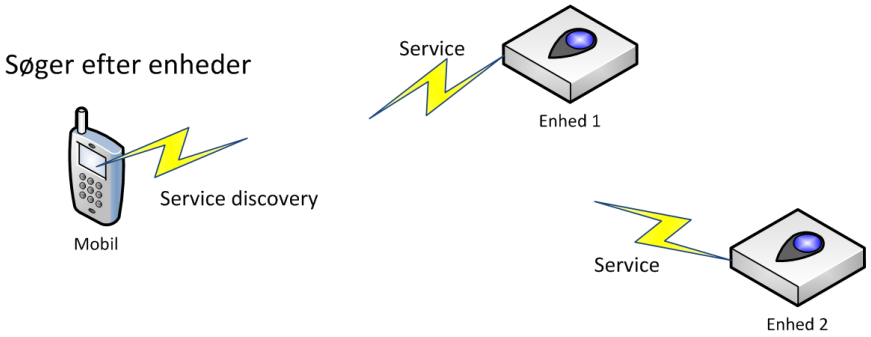
Frameworkets udformning er baseret på følgende ideer:

- De enkelte indlejrede enheder stiller en trådløs grænseflade til rådighed, der giver andre applikationer mulighed for at forbinde til dem. Enhederne optræder uafhængigt af hinanden og stiller deres grænseflade til rådighed for alle applikationer i nærheden.
  Grænsefladen er baseret på Bluetooth-kommunikationsprotokollen.

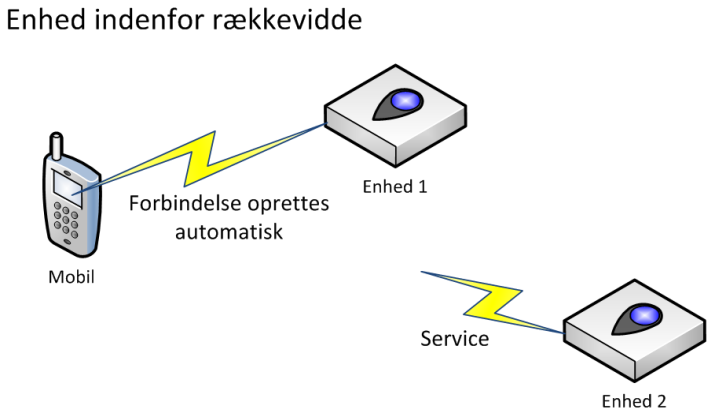


Figur 1: Trådløs grænseflade

- Det er muligt, for en mobilapplikation, automatisk at finde de indlejrede enheder når de er inden for rækkevidde (Figur 2). Dette sker igennem en service discovery protokol, der er implementeret både på mobilen og på de indlejrede enheder.
  Protokollen sørger for at enhederne er synlige og at mobilen kan forbinde trådløst til dem. På mobilen sørger protokollen for at applikationen konstant leder efter enheder og automatisk forbinder til disse, når de er inden for rækkevidde (Figur 3).



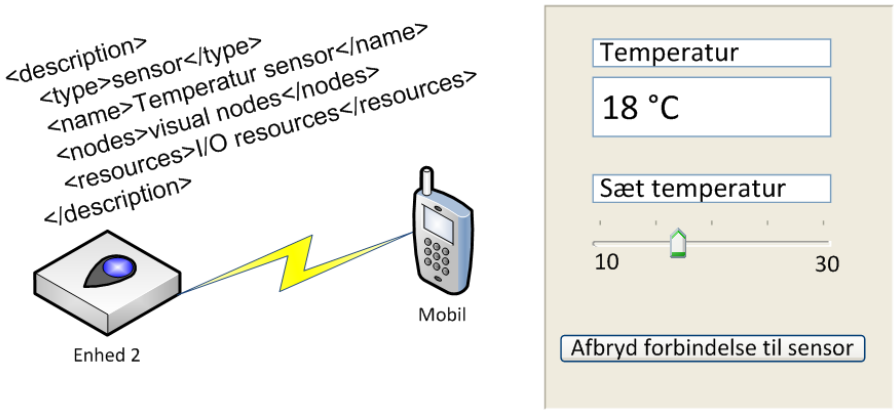
Figur 2: Søger efter enheder



Figur 3: Forbinder til enhed

- Den mobile applikation kan hente en beskrivelse af enheden og automatisk konfigurere brugergrænsefladen i forhold til denne.
  Beskrivelsen følger den protokol, der er defineret i *MOURDA*-platformen, uanset om det er en ny historie, en udvidelse af en eksisterende eller en uafhængig enhed.
  Herved er det muligt at benytte vilkårligt mange forskellige typer af indlejrede enheder, sensorer og lignende, så længe de enkelte enheder implementerer dette framework og deres indhold kan repræsenteres grafisk.

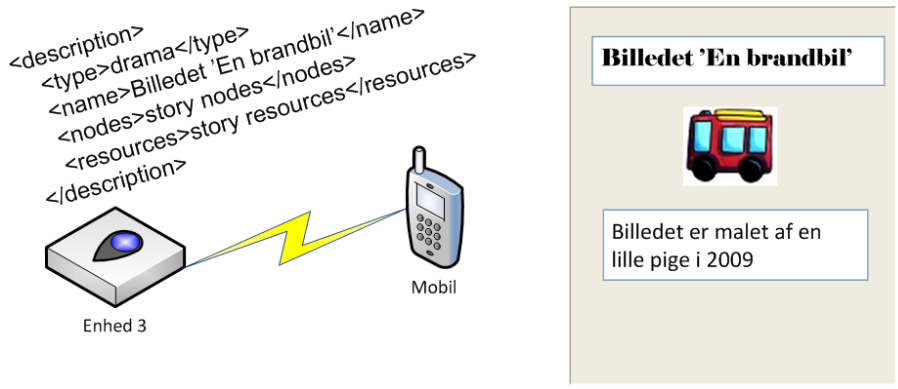
  Figur 4 viser hvordan en uafhængig sensorenhed sender sin beskrivelse til mobiltelefonen, der herefter viser et interface der matcher de muligheder der er i beskrivelsen (XML-beskrivelsen er ikke udspecificeret).



Figur 4: Forbundet til uafhængig enhed med sensorer

Figur 5 viser hvordan den første enhed sender en historie til mobiltelefonen. Brugeren får mulighed for at starte den.
Figur 6 viser hvordan den allerede påbegyndte historie bliver udvidet ved at hente en ny historie fra den næste enhed (f.eks. ved at brugeren bevæger sig rundt på et museum og kommer hen til et nyt billede).

Figur 5: Forbundet til enhed med historie



Figur 6: Udvidelse af eksisterende historie

Tilsammen giver disse elementer et generisk framework, der gør det muligt at implementere mobile applikationer, der kan udvides og konfigureres løbende.

1.5 Fremgangsmåde

Dette afsnit beskriver fremgangsmåden i projektet; hvilken metode der benyttes, hvilke trin der gennemgås, nødvendige ressourcer og en overordnet tidsplan med milepæle.

1.5.1 Metode

Dette afsnit beskriver gennemførelsen af projektet, der foregår gennem følgende trin. Hvert af hovedafsnittene afsluttes med en opsummering/evaluering af resultaterne.

Teoretisk undersøgelse

Første trin vil være en teoretisk undersøgelse af de resultater, der tidligere er blevet fundet i forbindelse med tilsvarende projekter, samt en sammenligning af disse med dette projekts formål og afgrænsning. Formålet med trinnet at finde ud af hvor langt forskningen og det praktiske arbejde, med at implementere sådanne løsninger, er, samt at få en viden omkring de teoretisk muligheder indenfor emnet.

Hovedvægten bliver lagt på at undersøge forskellige typer af dynamisk udvidelse af digitale historier, mobil sensorinteraktion i historier og automatisk generering af brugergrænseflader.

Herudover skal det undersøges hvorledes automatisk detektering (service discovery) af indlejrede enheder fungerer, samt hvordan en mobil applikation kan konfigureres til automatisk at forbinde til eksterne enheder.

Derudover skal Arduino-platformen, og tidligere arbejde med denne, undersøges, for at kunne drage nytte af eksisterende viden omkring implementering på platformen

Alt sammen har til formål at ligge til grund for en praktisk implementering af frameworket.

Mobilapplikation og konfigurationsprotokol

Formålet med dette trin er at få implementeret en konfigurationsprotokol, som en udvidelse af *MOURDA*-platformens grafiske beskrivelse, der kan benyttes til at konfigurere mobilapplikationen i forhold til en indlejret enheder. Denne protokol skal beskrive de funktioner enheden stiller til rådighed, eller beskrive de dele af den interaktive historie som enheden har ansvaret for.

Protokollen er den, der i afsnit 1.4 *Frameworkbeskrivelse*, blev brugt til at beskrive hvordan en ekstern enhed kan konfigurere den mobile applikation.

Herudover er formålet at få udvidet selve *MOURDA*-platformen, så den kan konfigureres og udvides dynamisk, ved at modtage nye beskrivelser fra eksterne enheder.

Konfigurationsprotokollen implementeres ved at lave en modificeret version af *MOURDA*-platformens XML-beskrivelse af grafik og historie.

Der skal tilføjes elementer til XML-beskrivelsen, der kan benyttes til at beskrive kommunikation med eksterne enheder, indsættelse og afvikling af nye historieelementer samt konfiguration af

sensorstyring. Samtidig skal de nødvendige kodekomponenter implementeres så de kan indsættes i en mobilapplikation. Applikationen skal kunne afvikles som en del af *MOURDA*-platformen.

Indlejrede enheder og kommunikationsprotokol

Dette trin omhandler udviklingen af den kommunikationsprotokol, som benyttes til at kommunikere i mellem mobilapplikationen og de eksterne enheder. Derudover skal den Bluetooth-baserede service discovery-protokol, som benyttes til at finde, og forbinde til, de indlejrede enheder også implementeres.

Service discovery-protokollen implementeres på både Arduino-enhederne og mobilapplikationen og den benyttes herefter på mobilen til at finde de eksterne enheder og forbinde til dem. Arduino-enhederne skal stille et velkendt interface til rådighed for mobiltelefonen.

Kommunikationsprotokollen implementeres ved at der laves en simpel protokol som gør det muligt for mobilapplikationen at forespørge efter XML-beskrivelsen og foretage simple aflæsninger/ændringer på de indlejrede enheder. Denne protokol benyttes efter at der er blevet oprettet forbindelse imellem mobilapplikationen og den indlejrede enhed.

Herudover skal der implementeres et softwareframework til Arduino-platformen, som gør det muligt at implementere de eksterne enheder på en simpel og konsistent måde, så de overholder de nødvendige protokoller.

Prototype og verifikation

Efter at have implementeret den grafiske protokol, kommunikationsprotokollen samt softwaren på Arduino-enhederne, skal der implementeres en proof-of-concept prototype, der kan benyttes til at evaluere frameworket og verificere at de teoretiske overvejelser og ønsker kan implementeres i praksis.

Der oprettes en simpel prototypehistorie, som indeholder både historieudvidelse og fysisk interaktion. Historien implementeres i en mobilapplikation og en række indlejrede enheder (Arduino-enheder) der forbindes til fysiske sensorer/aktuatorer.

Prototypen sammenlignes med den teoretiske viden, der blev opnået i den teoretiske undersøgelse, samt med de krav, der stilles til den praktiske implementering, jævnfør projektets formål og afgrænsning.

Evaluering og refleksion

Det sidste trin er en generel evaluering af hvorvidt projektet har opnået målet, samt hvorvidt den fundne løsning vil kunne bruges i praktiske implementeringer af dynamiske interaktive historier.

Samtidig vil der blive reflekteret over de forskellige afsnits resultater og deres relevans i forhold til projektets teoretiske baggrund.

1.5.2 Fysiske ressourcer

For at projektet kan gennemføres er der en række fysiske ressourcer, der skal være til rådighed. Følgende softwareressourcer skal være til rådighed.

- Arduino-framework til sensorudvikling og udvikling af kommunikationsprotokol
- Java ME udviklingsværktøj til mobiltelefoner

Følgende hardwareressourcer skal være til rådighed.

- Forskellige Arduino-enheder
- En række sensorer/aktuatorer til de fysiske installationer
- Mobiltelefon med Java ME og Bluetooth

1.5.3 Vejledere og samarbejdspartnere

Projektet gennemføres i samarbejde med følgende vejledere og samarbejdspartnere.

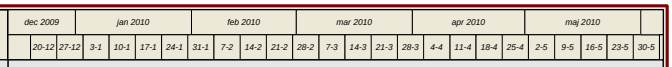
- Frank Allan Hansen, projektvejleder
- Kaj Grønbæk, Aarhus Universitet, hovedvejleder
- Alexandra Instituttet A/S

1.5.4 Tidsplan med milepæle

Projektet forventes afviklet jævnfør følgende Gannt-diagram.

Der er indlagt 4 milepæle i projektets afvikling, et efter hvert af de forskellige hovedafsnit.

	Opgavenavn	Start	Slut	Længde
1	**Teoretisk undersøgelse**	**14-12-2009**	**08-01-2010**	**20d**
2	Eksisterende teknologi	14-12-2009	01-01-2010	15d
3	Evaluering	04-01-2010	08-01-2010	5d
4	**Milepæl: Teoretisk projektbeskrivelse er klar**	11-01-2010	11-01-2010	0d
5	**Mobilapplikation og konfigurationsprotokol**	**11-01-2010**	**19-03-2010**	**50d**
6	Implementering	11-01-2010	12-03-2010	45d
7	Evaluering	15-03-2010	19-03-2010	5d
8	**Indlejrede enheder og Bluetooth-protokol**	**01-02-2010**	**05-03-2010**	**25d**
9	Implementering	01-02-2010	01-03-2010	21d
10	Evaluering	01-03-2010	05-03-2010	5d
11	**Milepæl: Framework implementering færdig**	22-03-2010	22-03-2010	0d
12	**Prototype**	**22-03-2010**	**09-04-2010**	**15d**
13	Implementering	22-03-2010	02-04-2010	10d
14	Test og evaluering	05-04-2010	09-04-2010	5d
15	**Milepæl: Proof-of-concept færdig**	12-04-2010	12-04-2010	0d
16	**Afslutning**	**12-04-2010**	**28-05-2010**	**35d**
17	Rapport	12-04-2010	14-05-2010	25d
18	Evaluering og reflektion	17-05-2010	21-05-2010	5d
19	Konklusion	24-05-2010	28-05-2010	5d
20	**Milepæl: Aflevering af speciale**	04-06-2010	04-06-2010	0d



Figur 7: Tidsplan

1.6 Resultat

Efter gennemførelse af projektet, er følgende er blevet implementeret og evalueret.

A. Et generisk softwareframework der opfylder projektets formål, som er at kunne udvide en interaktiv historie dynamisk igennem interaktion med indlejrede enheder, samt at interagere med fysiske sensorer/aktuatorer via trådløse kommunikation til indlejrede enheder.
   Følgende elementer i frameworket skal være implementeret:
   - A1. En softwareprotokol til at håndtere service discovery af de indlejrede enheder.
   - A2. En protokol til kommunikation mellem mobilapplikationen og de indlejrede enheder.
   - A3. Et interface til at aflæse og styre sensorer/aktuatorer, der er koblet til de indlejrede enheder.
   - A4. Et interface til dynamisk at udvide en eksisterende historie, på mobiltelefonen, ved hjælp af en beskrivelse hentet fra en indlejret enhed.

B. En proof-of-concept prototype, der implementerer softwareframeworket, med følgende elementer:
   - B1. Det skal være muligt at starte en historie og udvide denne, ved at interagere med indlejrede enheder.
   - B2. Det skal være muligt at interagere med fysiske sensorer og aktuatorer, via en igangværende historie.

C. En diskussion af frameworkets tilstand og dets forventede effekt på udviklingen af fremtidige interaktive historier.

Kapitel 2. Relateret arbejde

Dette speciale omhandler integrationen mellem digitale interaktive historier og fysisk interaktion med den verden som historien afvikles i. En række relevante projekter bliver herunder undersøgt og de metoder de benytter, til at løse eller behandle deres problemstillinger, bliver undersøgt og sammenlignet med de hovedemner, som dette speciale handler om:

- Interaktionsmetoder i digitale interaktive historier
- Automatisk generering af brugergrænseflader
- Discovery af eksterne services fra en mobil enhed
- Dynamisk udvidelse af interaktive historier
- Fysisk interaktion med eksterne enheder

2.1 Relaterede projekter

Interaktive historieplatforme til mobiltelefoner er blevet undersøgt og implementeret i en række relaterede projekter (Hansen, Kortbek & Grønbæk 2008, Scheible, Tuulos & Ojala 2007, Reitmaier, Marsden 2009). En del af disse undersøgelser omhandler også integrationen mellem direkte fysisk interaktion og den digitale historie (Gustafsson et al. 2006, Bichard et al. 2006, Kurdyukova, André & Leichtenstern 2009, Leichtenstern, André & Vogt 2007).

De to emner, interaktive historier og fysisk interaktion, er blevet undersøgt hver for sig, men sammenkoblingen mellem dem har vist sig at være vanskelig. Fysisk interaktion giver en bruger mange muligheder for at interagere med historien, og mange af disse muligheder kan være vanskelige at tage højde for på forhånd.

Tidligere projekter, der har arbejdet med sammenkoblingen af disse elementer, vil herunder blive undersøgt og diskuteret. Der ses også på tidligere forsøg med at udvikle generiske frameworks til fysisk interaktion via mobiltelefoner.

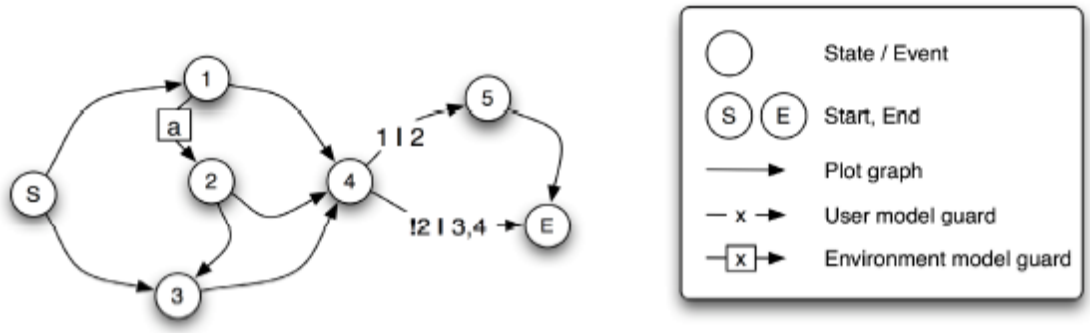
2.1.1 MOBILE URBAN DRAMA

Mobile Urban Drama (MOURDA) (Hansen, Kortbek & Grønbæk 2008) er et forskningsprojekt, der inddrager det fysiske urbane miljø i en interaktiv historiefortælling.

I systemet opbygges en fortælling ved hjælp af multimedier (billeder og lyd(narrativ)) som inddrager brugeren direkte i historien. Ved at træffe forskellige valg i historien kan brugeren påvirke historiens udvikling og føle at hun er en aktiv del af historien. Historien afvikles på mobiltelefoner der giver mulighed for at afspille mediefiler og vise billeder. Derudover giver mobiletelefonerne mulighed for at tage billeder af grafiske tags vha. det indbyggede kamera. Interaktion med det fysiske miljø håndteres ved at scanne disse tags og afspille de korrekte lydklip.

Det er også muligt at benytte telefonens indbyggede GPS-modtager til at bestemme brugerens lokation og ændre historieafvikling ud fra denne (for telefoner uden GPS kan scanning af tags også benyttes, da tags bliver hængt op på kendte lokationer).

Historier i MOURDA opbygges som en række tilstande/events hvor interaktioner og valg bestemmes af transitioner mellem dem (og eventuelle guards på transistionerne til at bestemme branching).



Figur 8: Tilstande/events i MOURDA

Et ekstra lag af interaktion kan indbygges ved at benytte sig af mobiltelefonens andre muligheder (SMS'er, telefonopkald og lign.).

2.1.2 ORIENT

ORIENT-projektet (Kurdyukova, André & Leichtenstern 2009) er et undersøgelsesprojekt, der omhandler inddragelse af multiple interaktionsmuligheder i en interaktiv historie. Systemet er målrettet mod skolebørn og historien deri opleves ved at børnene interagerer med en række fiktive væsner igennem fysiske bevægelser. Der benyttes både en Wii-controller, mobiltelefon med NFC og en elektronisk "dansemåtte", der kan registrere hvor man træder på den, til at interagere med den digitale verden i historien.

I *ORIENT*-projektet, finder de frem til en række retningslinjer for hvordan man succesfuldt føjer interaktion til en historie. Nogle af disse retningslinjer bliver diskuteret herunder.

Interaktionen med historien skal være transparent for brugeren.
Rådgivning og hjælp til interaktionen må ikke være påtrængende.
Den fysiske interaktion skal være en naturlig udvidelse af historien, så brugeren oplever den som et værktøj der kan benyttes til at skabe fremdrift og indlevelse i historien. Samtidig skal den give brugeren en indikation af hvor og hvornår interaktionen er mulig. Derved sikres at brugeren får nemmere ved at genkende og huske de interaktionsmuligheder, som en historie stiller til rådighed.

Samtidig må feedback til brugeren ikke være dominerede og påtrængende, men skal være integreret i historien, så brugeren oplever feedbacken som en del af historien, i stedet for som en del af en underliggende platform. F.eks. ved at feedback bygges ind i historien (en "guide" der følger med brugeren rundt) eller benytte de samme interaktionsmidler som er tilgængelige for brugeren.

Brugerens engagement kan øges ved at benytte velkendte interaktionsværktøjer.
Ved at bruge velkendte interaktionsmetoder, i projektet benytter de f.eks. en Wii-controller, kan man øge brugerens engagement og glæde ved at gennemgå historien. Samtidig bliver det nemmere og mere naturligt for brugeren at lære interaktionen at kende.

Typen af interaktion skal understøtte plottet/historien.
De interaktionsmetoder og -værktøjer man benytter til en historie, skal understøtte historien og kunne opfattes som en naturlig del af denne. Dette er vigtigt for ikke at bryde illusionen omkring historien og for at man kan benytte sig af selve plottet til at skabe en naturlig interaktion med den fysiske verden.

2.1.3 Approaches and Techniques to Achieve Dynamic Stories

I stedet for at udvide historien med nye elementer, er der blevet arbejdet på at lave systemer, som kan udvide en eksisterende historie ud fra brugerens valg og handlinger.

Et af disse systemer (Merabti et al. 2008) benytter sig af *Fuzzy Cognitive Goal Net* (Cai et al. 2006), til at opsætte nye mål og redefinere plottet, ud fra de handlinger som en bruger udfører. Mængden af mulige handlinger, mål, plotelementer, interaktioner osv. er en prædefineret mængde, som den intelligente *Goal Net* model så opererer på.

Ud fra et sæt regler kan historien dynamisk og intelligent blive omkonfigureret i forhold til input fra brugeren (eller andetsteds fra). Derved sikres at brugeren kan foretage en langt større mængde handlinger, og foretage sig valg som ikke med rimelig kunne forudses, uden at historien og plottet tabes.

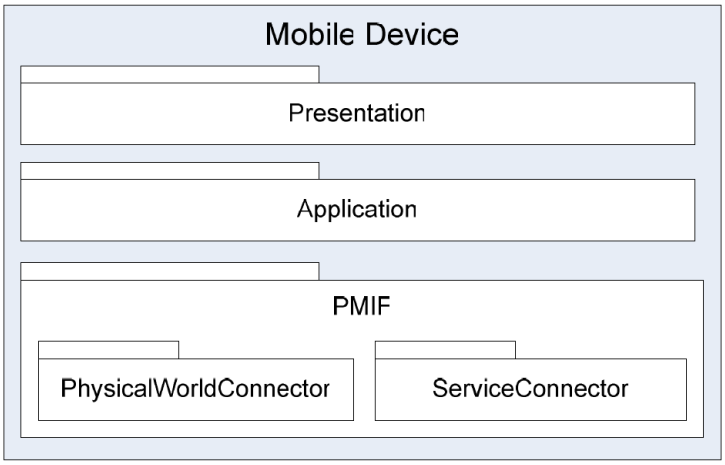
I stedet for kun at benytte brugerens bevidste valg til at konfigurere historien, kan man derfor benytte indirekte valg, baseret på de opsatte mål, som brugeren måske slet ikke ved, bliver truffet. Derved opnås en interaktiv historie, der for brugeren virker som om den er levende og som løbende konfigurerer sig i forhold til brugerens egen oplevelse.

Samtidig er det muligt at lade personerne, i historien, reagere på brugerens valg og derved skabe en levende verden med personer der har følelser og meninger omkring brugeren. Det vil gøre det muligt at lave en historie der ikke afvikles på samme måde hver gang og som samtidig har mulighed for, ved at brugeren skaber gode relationer til en karakter, at lave samarbejde mellem personerne i historien og brugeren.

2.1.4 Physical Mobile Interaction Framework

En tysk forskergruppe (Rukzio, Wetzstein & Schmidt 2005) har udviklet et generisk framework til at skabe interaktion mellem mobiltelefoner og mobile services. Frameworket, der kaldes *Physical Mobile Interaction Framework* (*PMIF*), fokuserer på mobil interaktion med den fysiske verden.

PMIF opstiller en lagdelt model, der gør det muligt for en mobilapplikation at kommunikere med vilkårlige eksterne objekter. Der fokuseres hovedsageligt på softwareimplementeringen af frameworket og det baserer sig på en række generelle idéer, hvor brugeren interagerer med mobiltelefonen, der så igen, igennem en række softwarekomponenter, kan interagere med den fysiske verden.

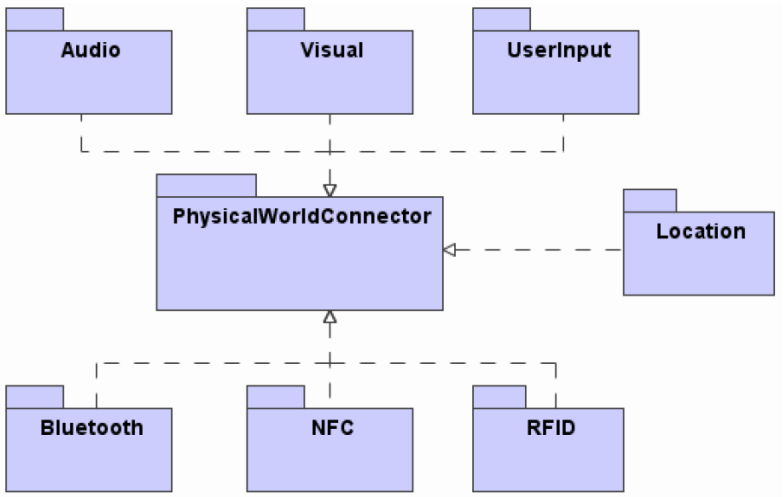


Figur 9: Lagdeling i PMIF

Frameworket baserer sig på en række trådløse kommunikationsteknologier, bl.a. Wi-Fi, Bluetooth og NFC, og benytter oftest kamera (grafiske tags) eller NFC til at skabe forbindelse med. Forbindelsen til eksterne objekter håndteres gennem et standard Java ME-Streaminterface , der muliggør tovejskommunikation (såfremt objektet i den anden ende, og kommunikationsprotokollen, understøtter dette).

Frameworket lægger op til at der benyttes web-services til at kommunikere ud fra telefonen, og at forbindelserne til disse services hentes gennem interaktion med et af de førnævnte eksterne objekter. Direkte fysisk interaktion med de eksterne objekter er ikke en prioriteret det af frameworket.

Eksterne objekter bliver abstraheret væk i PMIF og afkobles fra selve historiekoden i et *PhysicalWorldConnector*-objekt, for at sikre at selve den fysiske kommunikation ikke behøver at blive implementeret som en del af historien, men at historieudvikleren kan fokusere på interaktionsformen i stedet.



Figur 10: Generaliseret connector i PMIF

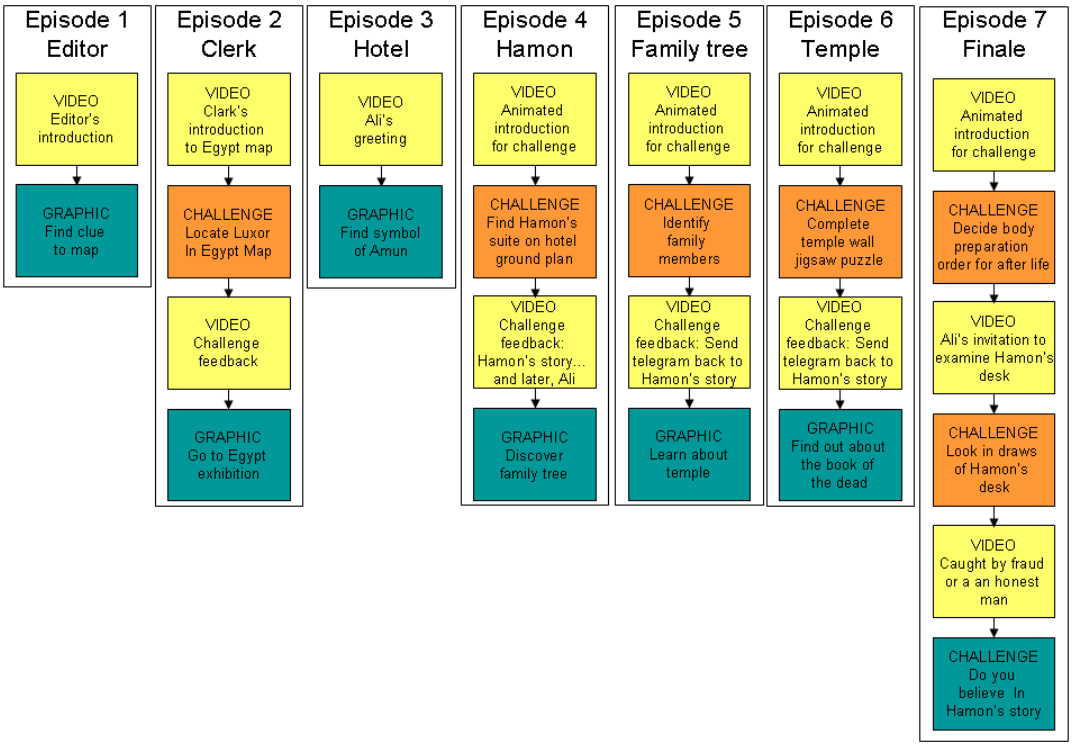
PMIF inddrager også et servermodul hvor kommunikation, interaktion og handlinger, i forbindelse med de eksterne objekter, kan lagres. Ligeledes kan serveren indeholde informationer omkring objekterne, som mobilapplikationen kan hente og /eller opdatere efterhånden som interaktionen foretages.

2.1.5 Interactive Storytelling Exhibition Project

Et Engelsk projekt omkring udvikling af interaktive museumsudstillinger, *Interactive Storytelling Exhibition Project* (*INSTEP*) (Danks et al. 2007), forsøger at kombinere interaktivitet og læring på Engelske museer. Projektet blev udviklet efter at undersøgelser havde vist, at gæsterne på museerne ønskede mere interaktion med udstillinger for at lære mere, og for at blive underholdt.

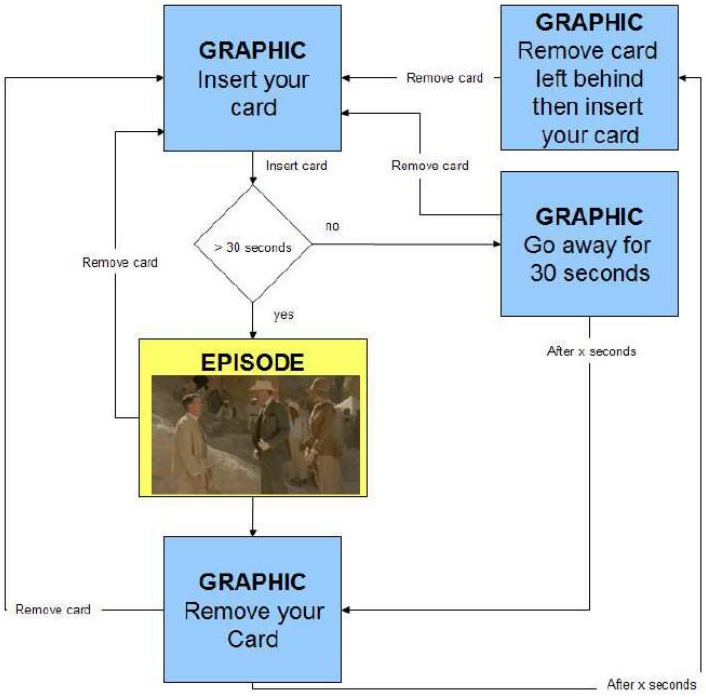
Brugen af it-systemer har været begrænset til stationære opstillinger enkelte steder på museet, eller de meget udbredte håndholdte "fortællere", som gæsterne kan låne og gå rundt med. De kan så høre om en bestemt udstilling ved at trykke på en knap når de kommer frem. I begge tilfælde er interaktionen på et minimum og det føles ikke som om oplevelsen er en del af museets udstilling. Samtidig er historierne meget lineære og ikke-interaktive.

INSTEP er et system, der afvikles på en lang række stationære opstillinger rundt om på museet. Gæsterne får udleveret et chipkort, som de kan benytte ved de forskellige installationer, for at få mere information (med video og lyd). Samtidig indeholder systemet en række spil, som gæsterne kan deltage i, ved de forskellige opstillinger. Gæsterne bliver sendt rundt til forskellige udstillinger på museet, for at løse forskellige opgaver der er relevante for den information de lige har fået og den udstilling de kommer hen til. Historien er delt op i en række episoder der hver har video, opgaver og grafik.



Figur 11: Opdeling af historien på museet

Den grundlæggende systemarkitektur baserer sig på en enkel idé, hvor gæsterne sætter deres kort i en station, får noget information, og derefter ikke kan vende tilbage til samme station indenfor et givent tidsrum (se Figur 12). Denne mekanisme er indbygget i historien, hvor gæsten er en journalist i 1920'ernes Ægypten. Derved "tvinges" gæsterne også væk fra installationerne og rundt i museet, uden at dette virker anmassende. Faktisk gik kun en meget begrænset del af den dårlige feedback på denne form for "lås" på historien.



Figur 12: Brugssekvens af opstilling på museet (med lås)

Igennem *INSTEP*-projektet opnås en større grad af interaktion med museet og dets udstillinger, samt en mere flydende og dynamisk historie, da gæsterne selv kan vælge hvilke stationer de vil besøge og i hvilken rækkefølge. De informationer gæsterne får ved de enkelte stationer, er afhængige af deres tidligere valg af stationer.

Gæsternes tilbagemeldinger var også generelt positive omkring den nye interaktionsform og kun ca. 15% havde problemer med at bruge systemet og den, for gæsterne, nye interaktionsform.

2.1.6 Mobile Service Usage through Physical Mobile Interaction

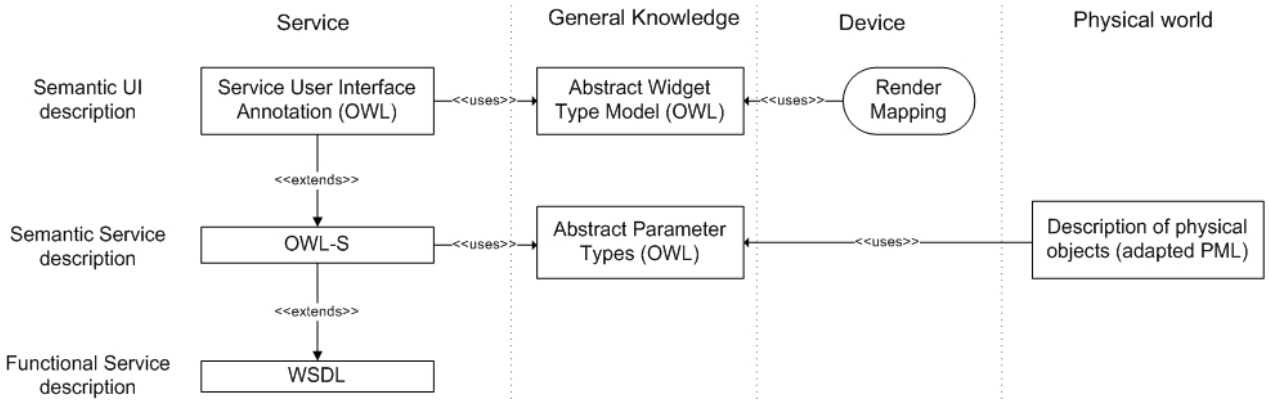
Et andet tysk forskningsprojekt har arbejdet videre med *PMIF* (2.1.4) og udvidet det med automatisk konfiguration og generering af en brugergrænseflade (Broll et al. 2007). I dette projekt fokuseres der på kommunikation med web services og benyttelsen af eksisterende servicebeskrivelsesprotokoller (f.eks. WSDL og OWL-S) som grænseflade- og kommunikationsbeskrivelse.

Idéen er at en bruger, igennem interaktion med en fysisk enhed (projektet har de samme interaktionsmuligheder som det tidligere projekt, f.eks. NFC eller kamera), automatisk opnår mulighed for at kommunikere med en web service, ved at de enkelte eksterne enheder giver mulighed for at hente en webservice-beskrivelse.

Brugergrænsefladen opbygges ud fra servicebeskrivelsen, som mappes over til en række grafiske elementer, der matcher de valg som servicen kræver/stiller til rådighed. Efter at brugeren har interageret med grænsefladen, benyttes den oprindelige servicebeskrivelse, og brugerens valg, til at generere et HTTP-request, der sendes til web servicen.

Igennem denne arkitektur er det således muligt, for en bruger, at interagere med en server, ved fysisk at interagere med eksterne enheder gennem f.eks. NFC eller Bluetooth.

Figur 13 viser hvordan en servicebeskrivelse benyttes til at genere UI og kommunikere med den fysiske verden.



Figur 13: Brugen af serviceinterface-beskrivelser i projektet

Brugerfeedback på prototypen, der blev udviklet og afprøvet i forbindelse med projektet, viste at brugerne af systemet havde relativt let ved at lære og bruge denne interaktionsform (i prototypen kunne brugerne købe billetter ved at interagere med en plakat). Dette galt både for NFC-berøring og ved scanning af grafiske tags.

2.1.7 Sensor-enhanced Mobile Web Clients

En forskningsgruppe, med tilknytning til Hewlett Packard Labs og en række Amerikanske universiteter, har arbejdet med at lave et projekt der omhandler generering af brugergrænseflader, automatisk detektering af services og sammenkobling af mobile enheder med disse services (Barton et al. 2003). Projektet hedder '*Sensor-enhanced Mobile Web Clients*' og benytter sig af modificerede HTTP-standarder til at finde og benytte sig af services.

Formålet med projektet var at finde en metode til service discovery for mobile enheder (med tilknyttede sensorenheder, f.eks. en mobiltelefon med et kamera), så de mobile enheder kan finde relevante services, der kan håndtere input fra de sensorenheder som mobilen har til rådighed. Metoden skulle virke begge veje, så den samtidig fungerede som en mulighed for services for at finde enheder, der stiller de informationer til rådighed som servicen kan bruge.

Service discovery har de implementeret ved at den mobile enhed kan sende en '*Produce*'-header til services, som beskriver hvilke medietyper mobilens sensorer kan generere (headeren er baseret på

opbygningen af *Accept*-headeren i HTTP[2], som benyttes til at angive hvilke medietyper en given service kan håndtere). Ud fra denne header kan servicen så returnere information om hvor en relevant service kan findes eller en beskrivelse af en brugergrænseflade til interaktion med den aktuelle service. Herved er det muligt for en mobil enhed at få fat på en service, der lige præcis kan håndtere den information som enheden kan producere (og ligeledes for servicen som kan aftage denne information). Efter discovery af en service opbygges der er brugergrænseflade, som gør det muligt at interagere med servicen.

GUI-beskrivelserne, der også benyttes til at kommunikation med servicen, benytter sig af XForms[3]. XForms er en XML-baseret beskrivelse som beskriver hvad en form kan og hvordan den ser ud. XForms afkobler data, logik og præsentation og de er derfor ikke en implementering af en grænseflade, men en formdefinition. Denne definition kan så vises vha. forskellige grænsefladeimplementeringer, f.eks. XHTML eller WML. Ved at benytte sig af forms, er datahåndtering og kommunikation allerede implementeret da dette er en del af formens indhold.

I projektet blev der også lavet en række udvidelser af XForms, som gør det muligt at sætte begrænsninger (constraints) på de enkelte inputfelter eller definere hvilke felter der skal vises, på forskellige platforme. Samtidig er det blevet muligt at binde sensorinput til et inputfelt, så mobilens sensorer automatisk udfylder de informationer, som servicen har brug for, når de aktiveres. Formen kan så enten sendes automatisk når sensoren har udfyldt felterne eller når brugeren ønsker det (afsendelse af formen angives også i beskrivelsen).

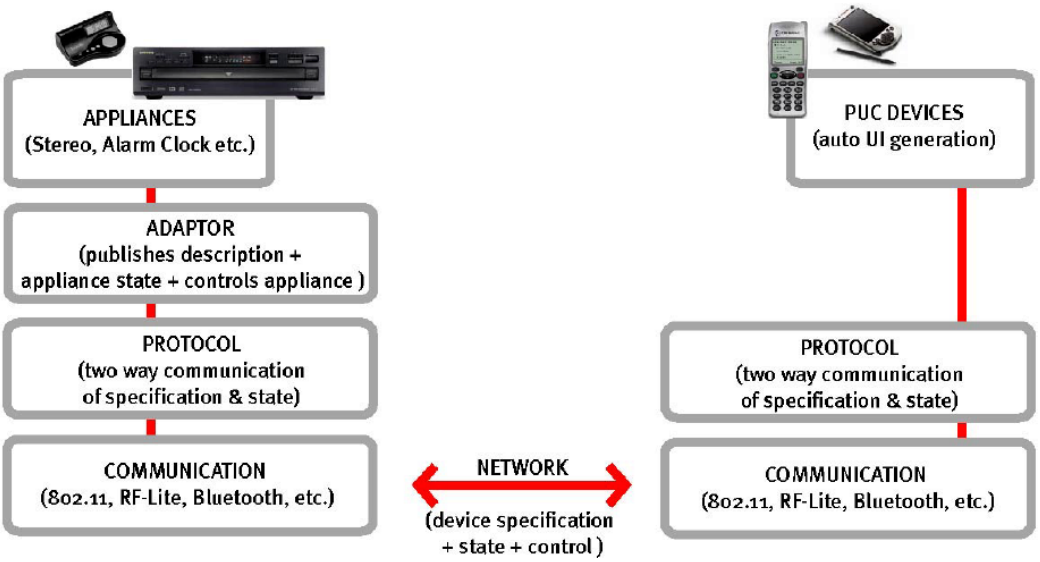
2.1.8 Personal Universal Controller

Et tidligere forskningsprojekt, der har arbejdet med automatisk at generere grafiske brugergrænseflader til apparater, er *PUC*-projektet (Nichols et al. 2002) fra Carnegie Mellon Universitetet.

Projektet handlede om at lave en interfacespecifikation, der kan bruges til at beskrive et apparats funktioner, i XML. Hvert apparat i systemet stiller så sin beskrivelse til rådighed og *PUC*'en kan så hente denne beskrivelse fra apparatet og generere en brugergrænseflade, der svarer til den. I *PUC* fokuseres der mest på brugergrænsefladen og der arbejdes en arkitektur op omkring den.

Arkitekturen gør det muligt, igennem forskellige lag, at kommunikere med apparater man kender beskrivelsen til (se Figur 14). Den benytter sig af adapterer til at kommunikere med de forskellige apparater, som hver især indeholder den kode, der er nødvendig for at foretage den fysiske kommunikation. Adapterne skal implementeres efterhånden som flere forskellige apparater kommer til, men de giver mulighed for at benytte sig af mange forskellige kommunikationsprotokoller; f.eks. FireWire eller Wi-Fi.

[2] http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html afsnit 14.1
[3] http://www.w3.org/MarkUp/Forms/#waXForms



Figur 14: Lagdeling i PUC

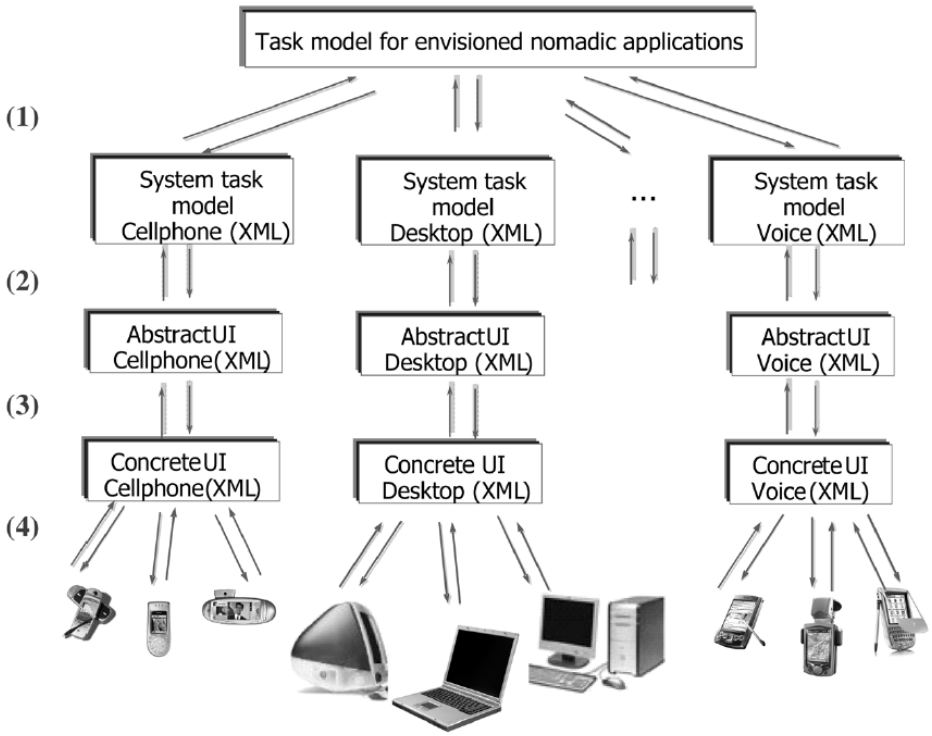
Den grundlæggende beskrivelse af *PUC*-interfacet er enkel og der benyttes som udgangspunkt kun to forskellige elementer til at beskrive apparatets funktioner; tilstande og kommandoer.
Tilstandene uddybes af en primitiv type (integer, boolean osv.) og en label. Kommandoer har kun en label. Ud fra disse to elementtyper kan brugergrænsefladen genereres og kommunikationen med apparatet fungerer ved at ændre værdier eller aktivere kommandoer.
Dette er en simpel, men effektiv, måde at beskrive en generisk brugerflade på. Ud fra ganske få grundelementer, kan de skabe brugerflader til vilkårligt mange eksterne enheder.

I projektet er service discovery delen abstraheret væk og det antages at der findes en generisk måde hvorpå *PUC*'en kan finde de enheder, den skal kontrollere.

Undersøgelser, foretaget i forbindelse med projektet, viste at brugerne lavede færre fejl og arbejdede hurtigere når de benyttede *PUC*'en til at interagere med deres apparater, sammenlignet med tilsvarende "universelle" fjernbetjeninger.

2.1.9 TERESA

Et Italiensk forskningsprojekt, kaldet *TERESA*, har arbejdet med at udvikle værktøjer til udvikling af applikationer til multiple platforme, ved at beskrive brugergrænsefladen og interaktionerne ved hjælp af logiske beskrivelser (Mori, Paterno & Santoro 2004). Grundidéen er at de benytter sig af en abstrakt beskrivelse af ønsket/mulig funktionalitet, som benyttes til at beskrive formålet med en given applikation. Denne beskrivelse transformeres så igennem en række lag, hvor hvert lag er mere konkret end det forrige, for til sidst at ende med en konkret brugergrænseflade. Den samme abstrakte beskrivelse kan benyttes på mange forskellige platforme (se Figur 15).



Figur 15: Lagdeling i TERESA

Første lag er en 'task and object'-model, der er en højniveau task-model som beskriver de opgaver (eller den funktionalitet) som applikationen skal udføre.

Næste lag mapper task-modellen til et abstrakt brugerinterface, som består af en række interactors (beskrivelser af interaktionskomponenter klassificeret i forhold til deres funktionalitet) og operatorer (der grupperer eller samler interactors i forhold til den ønskede præsentation). Interaktorerne beskriver f.eks. interaktioner som input, selections, editering eller lignende.

Tredje lag er en transformation fra en abstrakt brugerinterfacebeskrivelse til en konkret beskrivelse, der passer til den pågældende enhed. Her transformeres interaktorerne og operatorerne om til tilsvarende konkrete grafiske elementer på platformen (en 'select'-interaktor kan f.eks. blive mappet til en række radiobuttons, hvis det er et valg mellem forskellige muligheder, baggrundsbilleder kan blive aktiveret på en desktop men skjult på en mindre mobil enhed, osv). Herved opnås en grænsefladebeskrivelse af den oprindelige task-model, som er tilpasset den aktuelle enhed.

Sidste trin er generering af brugergrænsefladen på den aktuelle platform, ud fra den konkrete beskrivelse.

TERESA-projektets værktøjer gør det derfor muligt at lave en beskrivelse af ønsket funktionalitet på et højt niveau og ende ud med et automatisk genereret brugerinterface, som matcher den oprindelige beskrivelse i funktionalitet og interaktion.

2.2 Diskussion

De mange projekter belyser forskellige elementer af den samlede løsning, med forskellige måder at gribe problemerne an på og med varierende resultater. Men fælles for dem er, at de alle sammen er kommet med løsningsforslag der er relevante i forhold til dette speciales formål.

Herunder vil projekternes resultater og fremgangsmåde blive diskuteret, i forhold til de hovedemner der er nævnt i afsnittets begyndelse.

2.2.1 Fysisk interaktion og digitale interaktive historier

De forskellige projekter viser at det er uhyre kompliceret at lave et mobilt system, der interagerer med omverdenen på en måde som brugere opfatter som intuitivt. Selvom man forsøger at inddrage velkendte teknologier, som f.eks. mobiltelefoner, kamera og internet, er det ofte, for brugerne, ikke en velkendt form for interaktion man ender ud med. De er vant til at bruge telefonen som en telefon og ikke som et interaktionsmedie når de f.eks. skal købe billetter eller markere forskellige elementer.

INSTEP brugte stationære installationer og forsøgte at bygge en dynamisk historie op omkring dem, men det krævede at man tvang folk til at gå bestemte steder hen, i bestemte tidsrum, for at historien blev oplevet dynamisk. Dette er ikke nødvendigvis dårligt, jævnfør den generelt gode feedback, men det flytter fokus fra dynamikken i historien, til ”bare” at fortælle en historie og til at indføre multimedier nye steder.

Undersøgelserne viste at det er svært at beholde dynamikken i historien/dramaet, når man inddrager den fysiske verden. Oftest har man et specifikt mål og midlerne bruges så på at nå dette. Det giver god brugerfeedback, f.eks. i museer hvor gæsterne kan interagere med stationære computere, men dynamikken forsvinder. Brugerne har en vis grad af frihed, men mister mulighed for at opleve deres helt egen historie.

Dette gør sig gældende både for *INSTEP* og *PMIF*-udvidelsesprojektet, der begge viste at brugerfeedback generelt er positivt når der i miljøer, der normalt ikke havde interaktion, blev indført en ny interaktionsform. Mere interaktion medfører at brugere oplever historien som mere meningsfuld og de får en bedre indlevelse i historiens miljø.

2.2.2 Dynamiske historier og brugergrænseflader

To af projekterne fokuserer meget målrettet på brugergrænsefladen og dens repræsentation af det eksterne interaktionsobjekt. For begge projekterne gælder det, at de eksterne enheder har en XML-beskrivelse af deres funktionalitet som mobilapplikationen så kan hente og omsætte til en grafisk brugergrænseflade.

PUC-projektet handler specifikt om at skabe brugergrænseflader, ud fra en beskrivelse af et apparat, men der er ikke nogen historie. Det er stadigvæk en meget relevant undersøgelse og deres lagdelte arkitektur og primitivbaserede brugergrænseflade viser meget godt hvordan man kan omsætte en XML-beskrivelse til en funktionel interaktionsapplikation.

Dette vil også være relevant i historiebaserede scenarier hvor en brugergrænseflade skal skabes dynamisk. En XML-beskrivelse, der svarer til et givent punkt i historien, kan så enten genereres, aflæses eller hentes eksternt og direkte benyttes til at optegne en brugerflade i historien.

I *PMIF*-udvidelsesprojektet genereres der også brugergrænseflader ud fra en servicebeskrivelse, som samtidig kobles sammen med ekstern kommunikation og et drama.

Her kommer XML-beskrivelsen også fra en ekstern enhed, i samme stil som *PUC*-projektet, og det giver mange af de samme muligheder. At projektet så samtidig verificerer at beskrivelsen også kan benyttes til at skabe kommunikation med omverdenen, er bare et plus.

Samtidig var brugerfeedback på grænsefladerne generelt positivt og brugergrænsefladerne fungerede fint sammen med resten af interaktionen og historien.

2.2.3 Services og brugergrænsefladegenerering

XForms-projektet viser hvordan man kan benytte eksisterende protokoller og standarder, til at finde relevante services (eller producers) og kommunikere med dem. Ved at benytte XForms til at beskrive grænsefladen til servicen, er det muligt at sende en servicebeskrivelse til en mobil enhed uden at tage hensyn til hvordan den enkelte enhed har implementeret sin grafiske brugergrænseflade. Herved opnås en høj afkobling mellem servicebeskrivelsen og implementeringen af grænsefladen.

Formen oversættes til en standard som mobilen kan håndtere (f.eks. HTML), og det er så op til de enkelte enheder at generere en brugergrænseflade som indeholder de nødvendige elementer (i forhold til hvad mobilen kan producere). Og da XForms er en standardiseret protokol betyder dette at serviceimplementeringen kun behøver tage højde for funktionalitet, da det kan antages at enhederne, der modtager servicebeskrivelsen, kan håndtere visningen af formen. Samtidig giver brugen af forms mulighed for at sende resultatet (den udfyldte form) til en server (eller tilbage til servicen) vha. standard HTML-form håndtering.

Denne metode til at lave servicebeskrivelser minder om den metode, der benyttes i dette speciale. Begge metoder benytter sig af en funktionel XML-beskrivelse, der indeholder beskrivelser af input/output-muligheder, samt en beskrivelse af hvordan resultater føres tilbage til servicen. Forskellen ligger i brugen af forms, hvor XForms-projektet binder grænsefladebeskrivelse, service discovery og kommunikation op på (en udvidelse af) den standardiserede XForms-model. I dette speciale er grænsefladebeskrivelse og kommunikation delt op i to protokoller, der bindes sammen i en fælles service beskrivelse.

Brugen af XForms giver samtidig en række kontrolabstraktioner (select, input, output, trigger m.fl.) som gør det muligt at beskrive funktionalitet på brugergrænsefladen uden at beskrive implementeringen eller selve grænsefladen.

Et andet projekt, der omhandler brugergrænsefladegenerering (til mobile enheder) er *TERESA*-projektet. Projektet omhandler ikke mobile enheder specifikt, men metoderne de har undersøgt og implementeret gør at deres værktøj kan benyttes på mobile enheder.

Ved at starte med en abstrakt beskrivelse af brugergrænsefladen, baseret på abstrakte interaktionskomponenter (interactors), er det muligt at lave en applikation som kan vises, og interageres med, på mange forskellige platforme. Og ved at opdele transformationen i små trin, opnår de at det er muligt at benytte de samme transformationsmekanismer til at udvikle til flere forskellige platforme (omdannelsen fra task-model til en abstrakt interfacebeskrivelse er f.eks. fælles for alle platforme).

Ideen med at benytte abstrakte interaktionskomponenter i beskrivelsen, som blot angiver ønsket funktionalitet, og gruppere dem med operatorer, er en mulig udvidelse af den model der præsenteres i dette speciale. I stedet for at benytte konkrete interaktionskomponenter, der er tæt koblet med den fysiske præsentation, kan man indkapsle interaktionsmulighederne i mere højniveau elementer, som blot angiver hvad den ønskede funktionalitet er. Dette vil gøre det nemmere at udvide løsningen med flere muligheder, men også gøre det muligt at implementere den samme løsning på flere platforme.

2.2.4 Ekstern kommunikation og service discovery

Kommunikationen med den fysiske verden i projekterne, når historien er mobil, benytter oftest NFC, Bluetooth, kamera eller internet. Internettet fungerer bag grænsefladen og implementeres transparent for brugeren. Men det kræver at man benytter et andet medie til at igangsætte kommunikationen. NFC er populært, både blandt projekterne og blandt brugerne, men desværre er denne kommunikationsform stort set ikke-eksisterende i Europæiske telefoner pt. Bluetooth er ikke blevet undersøgt særligt meget, men enkelte af platformene lægger op til at understøtte det som interaktionsmedie. Kamera og grafiske tags er stadig en af de mest udbredte former for interaktion.

Nogle af projekterne har fokuseret en del på softwarearkitekturen omkring den eksterne kommunikation. Indkapsling af kommunikationskomponenterne, der står for at håndtere kommunikationen med eksterne enheder, er en vigtig del af arkitekturen. I *PMIF* benyttes en Connector-baseret arkitektur til at indkapsle alle de forskellige kommunikationsteknologier og de forskellige eksterne enheder, som systemet benytter sig af. Denne form for arkitektur er nem at udvide og afkoble, hvilket er en egenskab der udnyttes i det senere PMIF-baserede projekt (Broll et al. 2007), hvor den benyttes til at lave en serviceorienteret arkitektur til at kommunikere med web services.

Service discovery er et emne som de fleste af projekterne har abstraheret væk. Der antages enten at en protokol til discovery er til stede, eller der benyttes en alternativ metode til at finde services (f.eks. NFC eller kamera). Enkelte af projekterne har designet understøttelse for forskellige discovery mekanismer, men blot undladt at implementere det pga. tiden.

2.2.5 Sammenligning med specialets hovedemner

Specialets formål kan inddeles i fem hovedemner; interaktion, GUI-generering, service discovery, historieudvidelse og fysisk interaktion.

For at kunne sammenligne de ovenfor nævnte projekter med dette speciale, og omvendt, opsættes der en metrik, som kan benyttes til at foretage denne vurdering med. Metrikken kan ses nedenfor i Tabel 1.

Hovedemnerne er angivet i søjlerne og de forskellige projekters løsning/resultat på problemstillingen er angivet rækken ud for projektnavnet. Tomme felter angiver at projektet ikke omhandler emnet, eller kun i overfladisk grad.

Projekt	Interaktion	GUI-generering	Service discovery	Historieudvidelse	Fysiske enheder
ORIENT	Der opsættes et sæt af regler for design af interaktion.				
INSTEP	Interaktion håndteres igennem et kort, en række spil og multimedier. Brugeren sendes fysisk rundt i rummet.			Historien udvides igennem brugerens interaktion med udstillinger. Historien prædefineret men uden fastlagt kronologi.	Stationære opstillinger med tilkoblet audio/video. Der interageres vha. et udleveret kort.
PMIF	Via visuelle tags, RFID, NFC, Bluetooth eller andet interageres der med miljøet.		Visuelle tags, RFID, Bluetooth, mm.	Historien udvides igennem interaktion med miljøet.	Kommunikation igennem abstrakt PhysicalWorldConnector og et Java Stream-interface.
XForms	Post submit til en server/enhed via HTTP.	Høj niveau-beskrivelser af abstrakt funktionalitet. Generiske form-beskrivelser.	Beskrivelse af services/enhe der sendes via HTTP-headere.		Fysiske enheder mappes til inputfelter via en headerbeskrivelse.
TERESA		Lagdelt task-model med abstrakte interactors og operatorer. Samme beskrivelse benyttes på multiple platforme.			
PUC	Abstrakte kommandoer angiver ønsket interaktion med den aktuelle enhed.	Abstrakt XML-beskrivelse vha. tilstande og kommandoer.			Kommunikation håndteres via adapterer til de enkelte enheder og en standard transmissions-protokol.

Tabel 1: Metrik for relateret arbejde

Kapitel 3. Mobilapplikation og konfigurationsprotokol

Som nævnt i afgrænsningen i Kapitel 1 benyttes *MOURDA*-historieplatformen til implementeringen af den grafiske repræsentation og historiestyring. I dette kapitel bliver platformen analyseret og diskuteret, og det bliver undersøgt hvilke ændringer der skal foretages for at den passer ind i specialet.

Først beskrives *MOURDA*-platformen for at danne basis for resten af afsnittets analyser og konklusioner.

Herefter analyseres platformen for at finde ud af, hvad der evt. skal tilføjes eller ændres for at muliggøre de funktioner der undersøges i dette speciale. Heriblandt to af delspørgsmålene i specialets formål:

Delspørgsmål I (*Hvordan repræsenteres indlejrede sensorer og aktuatorer grafisk på en mobiltelefon?*)

Delspørgsmål II (*Hvordan skabes denne grafiske repræsentation dynamisk, uden at de indlejrede enheder og mobiltelefonen kender hinanden på forhånd?*).

Til sidst beskrives og implementeres den konfigurationsprotokol, der benyttes når de eksterne enheder sender deres funktionsbeskrivelse til mobilapplikationen.

3.1 Udvidelse af historieplatformen

For at kunne benytte historieplatformen MOURDA til dette projekt, er det nødvendigt at ændre og tilføje nogle elementer deri, for at tilgodese de nye muligheder der er udviklet i projektet.

MOURDA er stadig under udvikling og nogle af de elementer, der er nødvendige for dette speciale, udvikles som et naturligt led i frameworkets udvikling. Andre elementer tilføjes platformen som generelle udvidelser. I enkelte tilfælde kan det være nødvendigt at lave om på den måde hvorpå platformen fungerer, for at understøtte en ny funktionalitet.

3.1.1 MOURDA-platformens historieopbygning

Historiebeskrivelserne i *MOURDA* er opbygget som XML-beskrivelser, der indeholder alle de elementer som indgår i historien. XML-beskrivelsen består af en rodknude (’*drama*’), der indeholder fem andre hovedknuder, der så igen indeholder definerende underelementer.

```
<drama>
    <meta>…</meta>
    <properties>…</properties>
    <versions>…</versions>
    <resources>…</resources>
    <nodes>… </nodes>
</drama>
```

Eksempel 1: Drama XML

Meta

Denne knude indeholder en beskrivelse af historien.

```
<meta>
    <id>ny historieknude 1</id>
    <name>Den første nye knude</name>
</meta>
```

Eksempel 2: Meta properties

Properties

Denne knude indeholder en række underknuder (’*properties*’, der igen kan indeholder flere ’*property*’-knuder), som bruges til at sætte globale indstillinger for historien. Disse egenskaber bliver ikke benyttet under programafviklingen, men bliver under indlæsninger af historiebeskrivelsen indsat de steder, hvor de bliver refereret i andre elementer.

```
<properties>
    <property key="color">
        <property key="bgColor">FFFFFF</property>
    </property>

    <property key="color2" ref="color"></property>
</properties>
```

Eksempel 3: Drama properties

De enkelte properties, og andre elementer under historieknuderne, kan referere opad i dokumentet og automatisk få en reference oversat til indholdet af det XML-element, der refereres til (f.eks. bliver ’`color2`’ automatisk oversat til ’`FFFFFF`’ i eksemplet).

Versions

Her er det muligt at definere forskellige sprog, sprogspecifikke ressourcepakker og startpunkter.

```
<versions>
    <version startNode="main" resourceSet="dk" language="da">Dansk</version>
</versions>
```

Eksempel 4: Versioner

Ressources

URL’er og aliaser til alle historiens ressourcer. Downloades normalt under opstarten af programmet.

```
<resources baseUrl="http://exampleurl.com/res">
    <resource sizeInBytes="42436" filename="font.res" handle="font" />
</resources>
```

Eksempel 5: Ressourcer

Nodes

Selve historien. Den er inddelt i en række '*node*'-knuder, der hver har et id og som hver beskriver en grafisk komponent.

```
<nodes>
    <node id="main" component="ComponentTypeA">…</node>

    <node id="node2" component="ComponentTypeB">…</node>
</nodes>
```

Eksempel 6: Historieknuder

Under denne knude findes en række andre XML-elementer, der beskriver alle knudens indstillinger, moduler (interaktionsmuligheder) og de events der bliver genereret/kan håndteres.

```
<node id="main" component="ComponentTypeA">
    <properties>…</properties>
    <modules>…</modules>
    <eventhandlers>…</eventhandlers>
</node>
```

Eksempel 7: En enkelt historieknude

Historieflow

Flowet imellem knuderne håndteres igennem en event-mekanisme, hvor events enten sendes som følge af brugerinteraktion eller som implicitte events i forbindelse med systemhandlinger (musik der slutter, tid der er gået, osv.).

Figur 16: Eventhåndtering

Events håndteres af registrerede eventhandlers, der har mulighed for at påvirke komponenter eller moduler, eller starte en ny knude, ved at udføre den action der er tilknyttet et givent event.

Der er allerede defineret en række moduler, komponenter, events og actions i *MOURDA*.

Specielle properties

Et gennemgående element i XML-beskrivelsen er at tekststrenge angives med et id, som peger på en tekst der er angivet i en ekstern property-fil. Denne fil indeholder alle de tekstid'er, der benyttes i historien og de strenge id'erne skal erstattes med. Filen parses til et hash map, hvor id'erne er keys og strenge er values, som benyttes når der, på runtime, skal bruges en streng.

Denne metode til at angive tekststrenge med bliver benyttet af alle synlige elementer i applikationen.

Samtidig indeholder alle komponenter, og en del moduler, mulighed for at man kan angive en 'style'-property i elementet. Denne style benyttes til at vælge farver, layout og tema til elementet når dette er

synligt. Stylene er globale i beskrivelsen og er indeholdt i de generelle properties, og de bliver blot refereret fra de enkelte grafiske elementer.

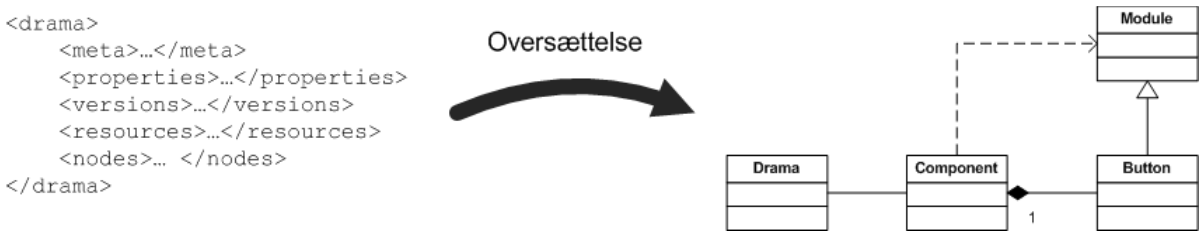
Ressourcehåndtering

Alle ressourcerne angivet under 'resources'-knuden downloades under første programstart, så de kan hentes lokalt under programafviklingen.

I samme stil som tekststrengene, angives eksterne ressourcer også med et id (eller handle) når de skal bruges i en komponent eller et modul. Dette id refererer til de aliaser der er defineret under 'resources'. Ud fra dette id kan en ressource hentes lokalt på runtime.

Afvikling af historien

XML-beskrivelsen indlæses under programopstarten og laves om til en objektmodel der gemmes i hukommelsen. Det er så denne objektmodel der benyttes under programafviklingen.



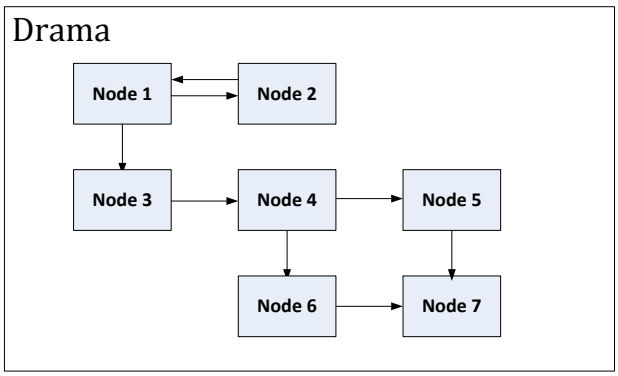
Figur 17: Oversættelse af historien til objektmodel

Derved sikres at afviklingen af historien kan foregå så hurtigt så muligt, men det betyder samtidig også at historieafviklingen er statisk, i det omfang at historien ligger fast både i forhold til start og slutpunkt.

3.1.2 Eksisterende arkitektur

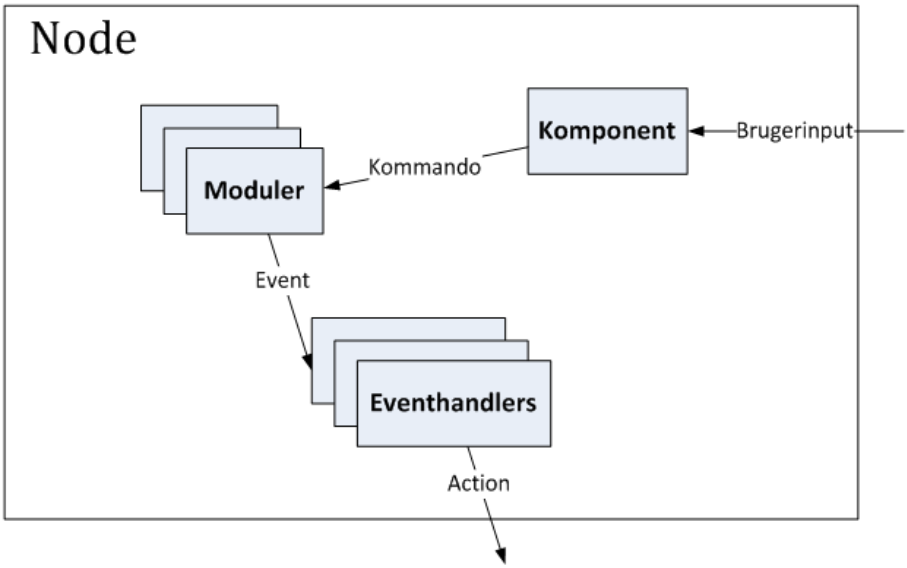
Den eksisterende arkitektur og det eksisterende historieflow i MOURDA-frameworket, er styret af de fælles relationer der er mellem knuder (med et id) og events/actions, der kan modificere eller starte dem.

Historien er opbygget som en flowstruktur, der giver mulighed for at historien kan bevæge sig imellem knuderne. Figur 18 viser hvordan flowet i en historie kan være. Her ses bort fra start- og slutelementer.



Figur 18: Flowdiagram over historie

De enkelte knuder indeholder moduler, eventhandlers og en grafisk komponent (se Figur 19 herunder), hvor komponenten vises til brugeren og dennes input håndteres af modulerne, der kan afsende events som håndteres af eventhandlerne. Disse events kan indeholde en action, der giver mulighed for at påvirke dramaet. Events kan også sendes fra moduler, uden at en kommando er blevet aktiveret af en bruger, f.eks. ved afslutning af et lydklip eller efter et specificeret tidsrum.



Figur 19: Hierarki i en knude

De actions der bliver afviklet, kan så enten påvirke de aktuelle moduler, den aktuelle komponent eller starte en ny knude.

3.1.3 Analyse af nødvendige tilføjelser

I forhold til platformens nuværende tilstand, er der tilføjet en række funktioner og muligheder i specialet. Jævnfør kravene til frameworket, beskrevet i afsnit 1.4 *Frameworkbeskrivelse* og kendskab til MOURDA-platformen, er følgende tilføjelser blevet implementeret.

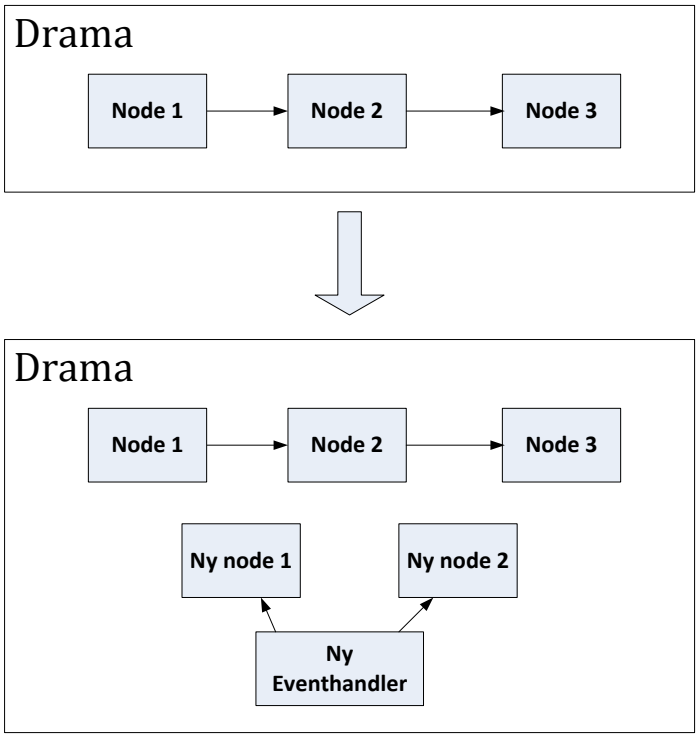
Mulighed for at føje nye knuder til historien på runtime

Historiebeskrivelsen (XML-filen) er som udgangspunkt en statisk historiestruktur, men efter at være blevet indlæst og lavet om til en objektstruktur i programmet, er det muligt at indføre eller ændre historieknuder.

Der er derfor udviklet funktionalitet, der gør det muligt at tilføje en eller flere nye knuder til en igangværende historie. Knuderne indlæses som XML-stumper, der parses på samme måde som hoveddramaet og indsættes i den samme objektstruktur. Derved bliver den nye knude automatisk en del af dramaet, dog uden at knuden umiddelbart kan tilgås da ingen events peger på den.

Mapningen mellem en knude og en igangværende historie, er det id knuden har, samt eventuelle eventhandlers der peger på dette id.

Derfor kan den nye funktionalitet automatisk indsætte eventhandlers, der peger på den nye knude, og som giver brugeren mulighed for at starte knuden. Eventuelt kan systemet automatisk sende events, der starter den nye knude op, alt afhængigt af den igangværende historie.



Figur 20: Tilføjelse af nye knuder

Nye grafiske komponenter

For at det er muligt at konfigurere en brugergrænseflade dynamisk, kræver det at de grafiske komponenter der er til rådighed, kan konfigureres i forhold til den funktionalitetsbeskrivelse der benyttes. Dette er et problem, da grafiske komponenter generelt er designet til at løse en specifik problemstilling og visualisere en specifik løsning (en 'Billede'-komponent kan f.eks. vise et eller flere billeder og en 'Spørgsmål'-komponent kan give brugeren et spørgsmål med svarmuligheder).

Derfor er der blevet føjet en dynamisk komponent til frameworket, hvor indholdet bestemmes ud fra primitiver og egenskaber i XML-beskrivelse. Primitiverne, der beskriver denne komponent, svarer til de funktioner en ekstern sensorenhed kan stille til rådighed. Funktionerne er generisk funktionalitet, som f.eks. at starte/stoppe en funktion, aflæse en værdi, sætte en værdi og lignende.

Samtidig er der ingen af de eksisterende moduler der giver mulighed for at repræsentere kommunikation med eksterne objekter. Frameworket er blevet udvidet med denne funktionalitet og moduler, der benytter sig af den, kan beskrives i XML'en.

Det er muligt for modulerne at fungere i baggrunden og ikke være direkte synlige for brugeren, hvis dette er nødvendigt for historien og typen af modulet. Kommunikationen mellem mobilapplikationen og den eksterne enhed svarer til de primitiver, som bruges til XML-beskrivelsen af den dynamiske komponent, der er nævnt ovenfor. Derved sikres at synlige moduler har en brugergrænseflade, der svarer til den eksterne enheds funktionalitet.

Nye moduler, der kan håndtere søgning efter, forbinde til og kommunikere med de eksterne enheder

Der er implementeret et kommunikationsmodul, der gør det muligt at lytte efter, og forbinde til, eksterne Bluetooth-enheder. Modulet kan afvikles i baggrunden og skal ikke være synligt i historien. Det detekterer, og forbinder, automatisk til de eksterne enheder der benytter dette framework. Efter at der er blevet oprettet forbindelse til en ekstern enhed, hentes og indlæses XML-beskrivelsen af denne.

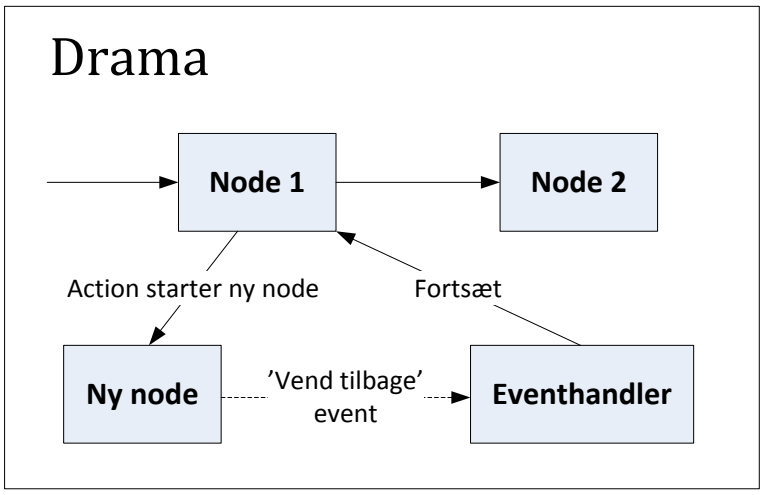
Samtidig er det muligt for andre historieelementer at abonnere på events, der fortæller når en ekstern enhed er fundet, samt at få kodemæssige håndtag (events eller id'er) til at kunne starte eller behandle den nye knude.

Ved at koble nye knuder sammen med historien via events, er det muligt via en global eventhandler, at håndtere nye knuder på samme måde igennem en hel historie. De kan enten bare tilføjes modellen, startes automatisk eller også kan brugeren spørges om hvad der skal ske. Alle mulighederne er tilføjet det eksisterende framework.

Historieflowet skal gøre det muligt at starte en ny knude og senere vende tilbage til den tidligere knude

I modsætning til den oprindelige historiebeskrivelse, hvor hele historien er kendt på forhånd, vil det i et drama, der benytter sig af interaktion med (på forhånd) ukendte indlejrede enheder, ikke være muligt for alle knuder at kunne vende tilbage til den oprindelige historie. De nye knuders eventhandlere kan ikke på forhånd vide, hvilke id'er der benyttes i historien, eller hvor langt i historien brugeren er kommet.

Derfor er frameworket blevet udvidet, så afviklingen af historien bliver logget og frameworket derved holder styr på hvor langt i historien en bruger er kommet. Det gør det muligt for en ny knude at afsende et event, der vender tilbage til den knude der var aktiv inden den eksterne enhed sendte sin beskrivelse. Derved opnås mulighed for at nye knuder kan indsættes på et vilkårligt tidspunkt, i en vilkårlig historie, uden at det oprindelige historieflow bliver ødelagt.



Figur 21: Nyt historieflow

3.1.4 Arkitekturændringer

Der er blevet foretaget en række ændringer af den eksisterende arkitektur, for at de nye tilføjelser kunne implementeres.

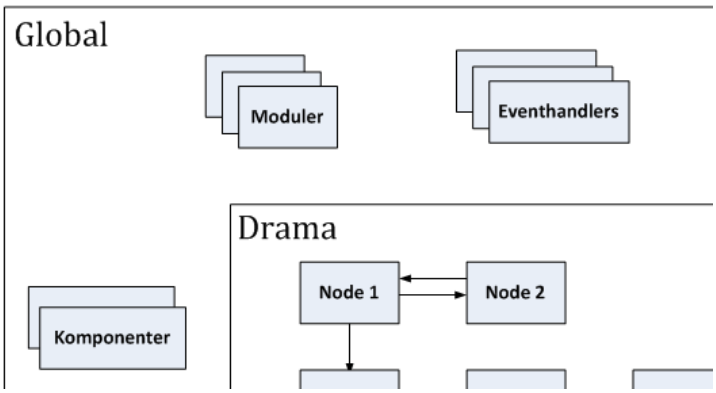
Globale elementer

For at kunne opsætte globale moduler, der lytter efter eksterne Bluetooth-enheder, er det nødvendigt at have moduler og eventhandlers der ikke lever inde i de enkelte knuder, men er aktive i selve applikationen udenfor historiens scope.

Denne ændring tilføjer et nyt scope til arkitekturen; et globalt scope, der lever udenom den aktuelle historie. I det globale område er det muligt at tilføje de samme elementer som findes i de enkelte knuder og det kan opfattes som en knude der altid er aktiv (se Figur 22). XML-beskrivelsen er blevet udvidet med en ny hovedknude '`global`', der har samme opbygning som de eksisterende 'node'-knuder.

Forskellen mellem de to typer af knuder er, at under indlæsningen af beskrivelsen bliver den globale knude flyttet udenfor historiens scope, og moduler og eventhandlers defineret deri afvikles i baggrunden.

Det globale område er ikke statisk og det kan konfigureres løbende på runtime, ved at nye knuder/dramaer selv indeholder en 'global' knude. Nye moduler, eventhandlers og komponenter bliver blot tilføjet det globale område under indlæsningen af den nye knude.



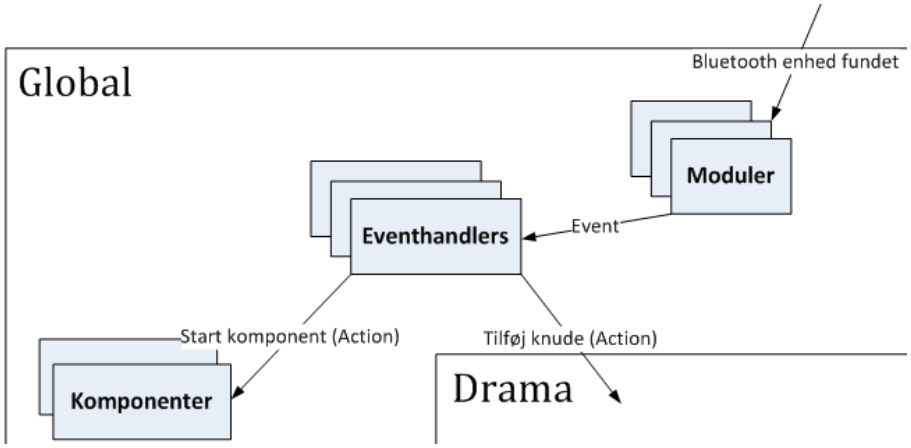
Figur 22: Globale elementer i dramaet

Eksterne enheder

Detektering af og kommunikation med de eksterne enheder, håndteres igennem de nye moduler i det globale område. Disse moduler lytter efter Bluetooth-enheder og forbinder automatisk til dem der er relevante.

XML-beskrivelsen af enheden hentes automatisk og modulet afsender et event, der indeholder denne beskrivelse. En eventhandler fanger eventet og udfører den ønskede handling, hvilket bl.a. inkluderer at indlæse beskrivelsen. Herefter tilføjes en knude til objektmodellen, en komponent aktiveres eller andre handlinger, der kan være relevante i forhold til beskrivelsen, udføres.

Figur 23 viser programflowet når en ekstern Bluetooth-enhed bliver fundet af systemet.



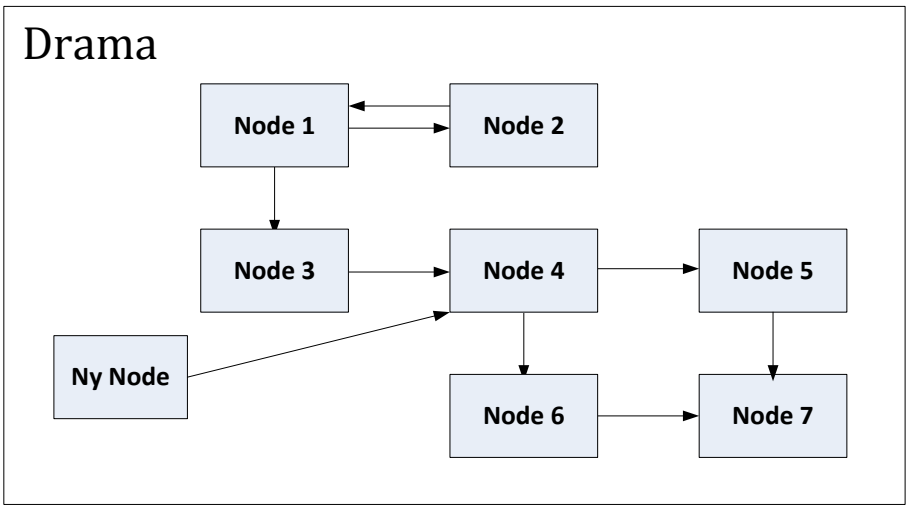
Figur 23: Håndtering af eksterne enheder

Nye knuder

I det tilfælde hvor en ny knude tilføjes historien, genereres denne knude på samme måde som de eksisterende knuder ud fra XML-beskrivelsen. Knuden indeholder standardelementerne; komponent, moduler og eventhandlers.

Hvis den eksterne enhed er en sensor/aktuator, eller der er mulighed for kommunikation mellem mobilapplikationen og enheden, benyttes den nye dynamiske komponent, som konfigureres via XML-beskrivelsen (se afsnit 3.2 *Konfigurationsprotokol* senere i dokumentet).

Efter oprettelsen bliver knuden tilføjet den aktuelle objektmodel og indgår, på lige fod med de eksisterende knuder, i historien. Komponenten kan startes så eventhandlers kan lytte på events og eventuelle transitioner til andre knuder bliver mulige. Figur 24 viser dramaets opbygning efter at en ny knude er blevet tilføjet, hvor denne knude har mulighed for en transition til 'Node 4', f.eks. via et event.

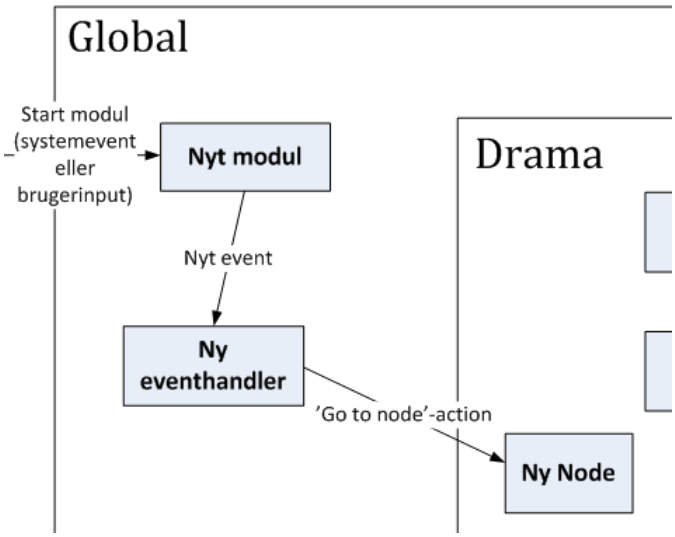


Figur 24: Ny knude tilføjet historien

Historieflow

Når en ny knude bliver tilføjet en igangværende historie, er det som udgangspunkt ikke muligt at aktivere knuden fra historien, eller at aktivere de eventhandlers den indeholder. Dette skyldes, som tidligere nævnt, at den nye knude ikke kender til den igangværende historie og omvendt.

Ved at benytte sig af den nye 'global' knude, er det muligt at beskrive historieknuden således at den har en eventhandler og et modul, der eksisterer udenfor dramaet. Modulet sender et event, der aktiverer den nye eventhandler, som så igen starter den nye knude. Modulet kan enten påvirkes af systemet eller af brugerinput (se Figur 25).



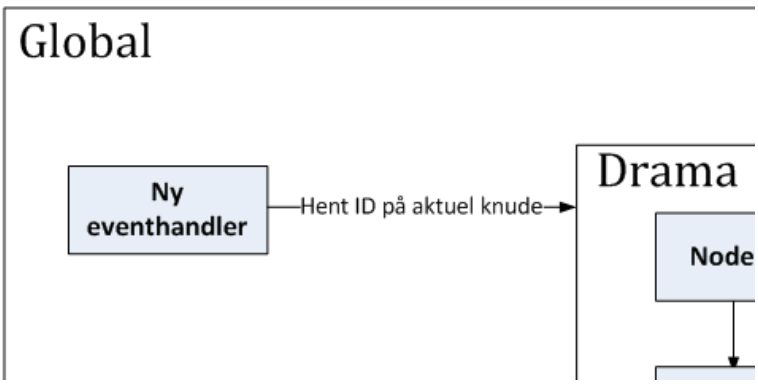
Figur 25: Start af ny knude

Herefter kan knuden afvikles som om den hele tiden har været en del af historien.

Tilbagevenden til historien

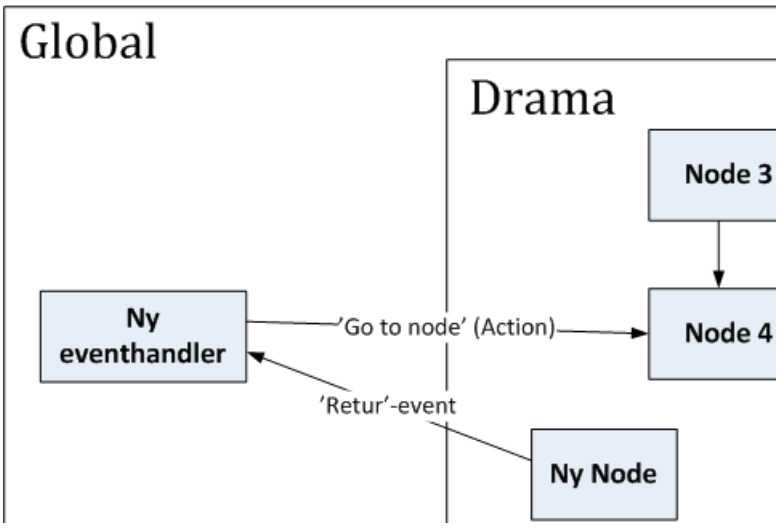
For at fortsætte historien er der to muligheder; enten genereres der et event, der starter en af de eksisterende knuder (dette forudsætter at den ny knude kender til historien eller eventuelt starter den forfra), eller også benyttes der et 'returevent' der håndteres som følger.

Samtidig med at komponenten bliver oprettet, oprettes der også en global eventhandler, som får en reference enten til den aktuelle eller den næste komponent, (f.eks. et id) fra det igangværende drama (Figur 26).



Figur 26: ID på aktuel knude hentes

Når den nye knude er færdig med at blive afviklet, sender den et event der fanges af den nævnte eventhandler, der så kan udføre en action som starter den rigtige knude.



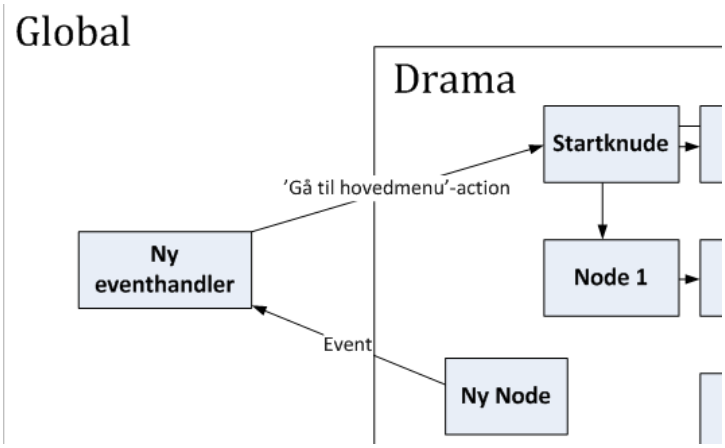
Figur 27: Returevent i aktion

Herefter kan det igangværende drama fortsætte som planlagt.

Retur-til-hovedmenu

Derudover er det muligt for en knude at benytte sig af en 'retur til hovedmenu'-action i en eventhandler. Denne action er tilgængelig for alle knuder, men retter sig specielt imod dynamisk genererede knuder, der indgår i dramaer hvor der ikke nødvendigvis er en aktuel knude eller et historieflow, men hvor de enkelte knuder er uafhængige elementer (f.eks. ved en rundvisning hvor de enkelte poster er uafhængige elementer, der ikke kædes sammen i en historie).

Actionen starter den startknude, der er specificeret i den XML-fil, der hører til den aktuelle historie.



Figur 28: Retur til hovedmenu

Denne aktion giver dynamiske knuder mulighed for at vende tilbage til en hovedmenu, uden at de skal være afhængige af tilstedeværelsen af eksterne eventhandlers, da aktionen kan pakkes ind i en eventhandler som er indeholdt i selve knuden.

Indlæsning af ressourcer

Da de nye knuder fra en ekstern enhed kan være afhængige af ressourcer, der muligvis ikke er blevet indlæst i programmets oprindelige historie, skal det være muligt at downloade disse ressourcer når nye knuder har brug for det.

For at håndtere denne situation, tilføjes der en ny aktion, som kan eksekveres fra eventhandlers. Denne aktion ('LoadGlobalResourcesAction') sikrer at de globale ressourcer, der er angivet som properties til aktionen, er downloadet, og hvis de ikke er, bliver de downloadet automatisk.

Eksempel 8 viser opbygningen af den XML, der beskriver den nye aktion.

```
<action name='net.interactivespaces.mud.actions.LoadGlobalResourcesAction'>
    <properties>
        <property key='resources'>
            <property key='1'>gogh1</property>
            <property key='2'>gogh2</property>
            <property key='3'>gogh3</property>
        </property>
    </properties>
</action>
```

Eksempel 8: Download af nye ressourcer

3.2 Konfigurationsprotokol

I dette afsnit beskrives den protokol der benyttes til at konfigurere den dynamiske historie, når der kommunikeres med en ekstern enhed, og den dynamiske komponent der håndterer den grafiske repræsentation af denne enhed.

Først beskrives den generelle protokol, hvorefter de to forskellige interaktionsmuligheder; en ny knude til historien og en interaktiv sensor-/aktuatorenhed, beskrives.

Herefter designes komponentens opbygning og konfiguration i forhold til XML-beskrivelsen.

3.2.1 Analyse af protokol

Den generelle opbygning af konfigurationsprotokollen er magen til den historiebeskrivelsesprotokol, der findes i MOURDA-frameworket. Konfigurationsbeskrivelsen indeholder derfor de samme elementer og muligheder, som den grafiske repræsentation muliggør.

Konfigurationen fungerer ved at den XML-beskrivelse, der modtages fra en ekstern enhed, enten er en hel dramabeskrivelse (hvis der er behov for style og ressourcer) eller en beskrivelse af en enkelt uafhængig knude (hvis dette er nok). For en gennemgang af XML-beskrivelsen, se afsnit 3.1.1 *MOURDA-platformens historieopbygning*.

Dog er der blevet tilføjet flere typer af standardelementerne (komponenter, moduler og eventhandlers), for at sikre at de eksternt hentede beskrivelser kan indlæses og afvikles i applikationen. Herunder beskrives de ændringer i protokollen, der er blevet foretaget for at implementere de to mulige typer af konfiguration, samt de nye klasser der er blevet tilføjet systemet.

Nye konfigurationselementer

For at kunne håndtere nye knuder korrekt, er det nødvendigt at deres beskrivelse indeholder de elementer der benyttes til at konfigurere og starte dem. Disse konfigurationselementer er XML-knuder der allerede findes i et eksisterende drama, men som benyttes her til at beskrive hvorledes de nye knuder skal sættes op. Følgende elementer skal være indeholdt i XML-beskrivelsen for en ekstern enhed.

```
<interactive>
  <properties>
    ...
  </properties>
  <eventhandlers>
    ...
  </eventhandlers>
  <resources baseUrl='URL til ressourcer'>
    ...
  </resources>
  <nodes>
    ...
  </nodes>
</interactive >
```

Eksempel 9: Elementer i beskrivelsen af en ekstern enhed

De enkelte elementer beskrives herunder.

'interactive'-knuden er blot en rodknude der samler de forskellige elementer.

Properties

'properties'-knuden benyttes til at angive værdier, der benyttes under opsætning og opstart af knuden. Hvis der ikke er properties at sætte skal knuden være tom.

Der er implementeret en enkelt prædefineret property key, som kan benyttes (andre keys er også mulige, denne key har bare en bestemt betydning):

discoveryQuestion – der er et spørgsmål, som frameworket kan stille brugeren før den nye enheds knuder bliver startet. Hvis propertien udelades stilles der intet spørgsmål. For eksempel:

```
<property key='discoveryQuestion'>
  Ønsker du at se flere værker af Van Gogh?
</property>
```

Eksempel 10: DiscoveryQuestion til en ekstern enhed

Eventhandlers

Denne knude beskriver de globale eventhandlers, som skal sættes op før knuden kan fungere. Det er muligt at sætte alle gængse MOURDA eventhandlers op her, men der skal som minimum sættes en eventhandler op der håndterer et 'onload'-event.

Denne event sendes automatisk hvis brugeren svarer ja til det ovenstående 'disovery question' eller hvis spørgsmålet er udeladt.

```
<handler event='net.interactivespaces.mud.events.OnLoadEventHandler'>
    <guard>
        …
    </guard>
    <action name='…'>
        …
    </action>
</handler>
```

Eksempel 11: Globale eventhandlers i en ekstern enhed

Guard'en på denne eventhandler skal sættes til navnet på den eksterne enhed, med filtypen '.xml', f.eks.

```
<guard>
    <properties>
        <property key='filename'>VanGogh.xml</property>
    </properties>
</guard>
```

Eksempel 12: Guard på OnLoadEvent

Actions kan være alle tilgængelige actions fra MOURDA og eventet gør det bl.a. muligt at sikre at der bliver håndteret en 'LoadGlobalResourcesAction' før selve komponenten bliver afviklet.

Resources

Denne knude er magen til en almindelig ressourceknude på et drama, dog bliver alle ressourcer, der er beskrevet heri, tilføjet den globale samling af ressourcer.

3.2.2 Konfiguration af historieknuder

Hvis den eksterne enhed konfigurerer en ny knude, der skal afvikles uden at der er kommunikation med den eksterne enhed, håndteres visningen og afviklingen af knuden igennem de allerede eksisterende historieelementer. Knuden opfattes som en uafhængig historieknude, der enten afvikles selvstændigt, eller som en del af en eksisterende historie.

Indlæsningen af knudens elementer kræver dog, at der er enkelte properties der er defineret i forhold til et eksisterende drama (Eksempel 13). Dette er style-properties og tekststrenge, som ikke nødvendigvis er defineret i den nye knude. Det er muligt at definere en style deri, men det er oftest ønskeligt at nye knuder minder om lignende knuder i det aktuelle drama.

```
<properties>
    <property key="titlekey"> postliste.title </property>
    <property key="textkey"> postliste.text </property>
    <property key="style" ref="instructionStyle"></property>
</properties>
```

Eksempel 13: Properties før

Derfor er der blevet foretaget en udvidelse af den eksisterende XML-beskrivelse (Eksempel 14), og indlæsningen af den, der medfører at følgende er muligt:

- Tekststrenge kan angives som komplette strenge, uden at det behøver være et id, der refererer historiens hash map af strenge (se afsnit 3.1.1 *Specielle properties*).
- Stylen af dynamiske komponenter kan referere en standard style til dynamiske enheder. Denne style '`dynamicStyle`' skal være implementeret i alle applikationer, der benytter frameworket.
- Eksterne ressourcer der benyttes af den nye komponent, hentes automatisk ned lokalt når knuden indlæses i applikationen. De id'er, der referer til de nye ressourcer, tilføjes den eksisterende liste af ressourceid'er, så komponenten kan referere og hente dem på normal vis.

```
<properties>
    <property key="titlekey"> Dette er titlen </property>
    <property key="textkey"> Her er indholdet </property>
    <property key="style" ref="dynamicStyle"></property>
</properties>
```

Eksempel 14: Properties efter

Gennem disse ændringer sikres det, at det er muligt at tilføje en ny historieknude til et vilkårligt drama.

Herefter indsættes knuden i historien, som beskrevet i afsnit 3.1.4 *Arkitekturændringer*.

3.2.3 Konfiguration af interaktive knuder

En interaktiv knude beskrives på samme måde som en historieknude, ved at beskrivelsen sendes som enten en 'drama'- eller en 'node'-knude, der indlæses vha. den almindelige parser. Men beskrivelsen adskiller sig ved at komponenten også skal definere de muligheder, som den eksterne enhed har.

Der findes mange forskellige typer af eksterne enheder, når der tales om interaktive knuder, og funktionalitetsbeskrivelsen skal kunne beskrive de muligheder som de enheder stiller til rådighed. Interaktion baseret på indlejrede enheder har dog fællestræk, nemlig at de fysisk benytter sig af sensorer og/eller aktuatorer til at udføre interaktionen. Hvordan disse komponenter interfacer med den indlejrede enhed, samt hvad deres specifikke funktion er, kan abstraheres væk i funktionalitetsbeskrivelsen ved at generalisere. Derfor opdeles funktionalitetsbeskrivelsen i to generelle dele; sensorer og aktuatorer. Disse to typer af generel funktionalitet dækker derfor over en stor del af de eksterne enheders interaktionsmuligheder. Konfigurationsprotokollen kan altid senere udvides med flere interaktionskomponenter, hvis dette bliver nødvendigt.

Fælles properties

Der er en række properties der skal sættes på alle interaktive knuder, uanset om det er en sensor eller en aktuator.

```
<property key='deviceName'>VanGogh</property>
<property key='title'> Van Goghs billeder </property>
<property key='textKey'> Her kan du se Van Goghs billeder. </property>
<property key='styleKey'>dynamicStyle</property>
```

Eksempel 15: Fælles properties for interaktive knuder

deviceName – er navnet på den eksterne enhed. Det skal angives for den enkelte komponent, så det er muligt for komponentens elementer at kommunikere med den eksterne enhed.

title og *textKey* – Er den titel og tekst, der skal vises på skærmen når komponenten er aktiv.

styleKey – er den style som komponenten benytter sig af. Det kan enten være en style, der er defineret i komponentens properties eller den globale 'dynamicStyle', der er defineret i det overliggende drama.

Aflæsninger (sensorer)

Når en ekstern enhed stiller en måling eller en aflæsning til rådighed, beskrives denne værdi, i XML'en, ved at angive den som en property under komponenten. Beskrivelsen indeholder de informationer, der er nødvendige for at mobilapplikationen kan aflæse enheden og vise målingen i brugerfladen. Den indeholder en række subproperties, der beskriver forskellige nødvendige egenskaber af sensoraflæsningen (se Eksempel 16).

```
<property key="sensors">
    <property key="sensorReadingX">
        <property key="name"> tempX </property>
        <property key="display"> Temperatur </property>
        <property key="type"> Bar </property>
        <property key="update"> auto </property>
        <property key="interval"> 10 </property>
        <property key="unit"> C </property>
        <property key="max"> 100 </property>
        <property key="min"> 0 </property>
        <property key="step"> integer </property>
    </property>
    ...
</property>
```

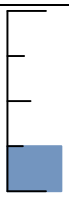
Eksempel 16: Sensoraflæsningsproperty

Sensoraflæsningsproperties har følgende opbygning:

- Yderst er property-grupperingen (med key='sensors'), der benyttes til at angive at alle efterfølgende properties beskriver eksterne sensorer.
- Hver enkelt sensor angives med en key, som benyttes internt i objektmodellen til at referere til objektet.
- Under de enkelte sensorer findes en række forskellige properties.
    - 'name' angiver et sensor id, der benyttes på den eksterne enhed. Dette id benyttes under kommunikationen med den eksterne enhed, til at få den rigtige aflæsning (se afsnit 4.3 *Kommunikationsprotokol* senere i dokumentet).
    - *'display'* er det navn, der skal vises i brugerfladen når værdien vises.
    - *'type'* angiver hvilken type visning der skal benyttes til at repræsentere værdien (se Tabel 2 herunder).
    - *'update'* angiver hvordan værdien af sensoren skal opdateres. Der er følgende tre tilladte værdier: none, manual og auto.
      'none' betyder at værdien ikke opdateres, 'manual' at brugeren manuelt skal opdatere den og 'auto' at værdien automatisk opdateres af systemet. Hvis 'auto' er sat, skal den næste property ('*interval*') også være sat.
    - *'interval'* er det tidsinterval (i sekunder) der går imellem automatiske opdateringer. En for kort tid her kan betyde at applikationen ikke laver andet end at opdatere sensorværdien. Hvis '*update*' ikke er sat til 'auto' har denne property ingen betydning og kan udelades.
    - *'unit'* fortæller hvilken enhed denne sensoraflæsning er i.
      (denne property er ikke påkrævet)
    - *'min/max'* er eventuelle minimum- og maksimumværdier for aflæsningen.
      (denne property er ikke påkrævet)

- *'step'* angiver de trin, hvormed værdien kan forandres (enten 'decimal' eller 'integer', afhængig af om værdien bruger decimaler eller ej).
  (denne property er ikke påkrævet)

I systemet er der blevet defineret en række sensorvisninger, som platformen implementerer. Disse typer benyttes til grafisk at repræsentere en sensor og dens aflæsning. I Tabel 2 herunder vises de forskellige typer der er tilgængelige, samt eksempler på deres visning.

Type	Eksempel
Simple Værdien vises som tekst i en kasse.	
Horizontal Værdien vises som en kasse, der fyldes op procentvis i forhold til min og max.	
Vertical Værdien vises som en søjle, der viser værdien i forhold til min og max.	

Tabel 2: Grafiske sensortyper

Ud fra disse properties kan mobilapplikationen, på runtime, indlæse en beskrivelse af en sensorenhed og grafisk repræsentere værdierne der aflæses derfra.

Påvirkning af den fysiske verden (aktuatorer)

Når en ekstern enhed stiller en aktuator til rådighed for historien, beskrives dette ligeledes i XML'en, ved at angive den som en property under komponenten (se Eksempel 17 hvor to forskellige typer af aktuatorer er vist). Denne property indeholder de informationer, der er nødvendige for at kommunikationen til den eksterne enhed kan foretages. Dog uden at beskrive kommunikationsprotokollen, der er indbygget i systemet og ikke er en del af de enkelte historier.

```
<property key="actuators">
    <property key="actuatorX">
        <property key="type"> state </property>
        <property key="name"> act1 </property>
        <property key="display"> Lys </property>
        <property key="states">
        <property key="1">Tænd</property>
        <property key="2">Sluk</property>
    </property>
    </property>
        <property key="actuatorY">
        <property key="type"> value </property>
        <property key="name"> act2 </property>
        <property key="display"> Temperatur </property>
        <property key="visualType"> Slider </property>
        <property key="max"> 100 </property>
        <property key="min"> 0 </property>
    </property>
</property>
```

Eksempel 17: Aktuatorproperties

Aktuatorenheder kan være en af to typer; tilstand eller værdi.
En 'tilstand'-aktuator (*state*) beskriver en aktuator, der har en eller flere tilstande den kan være i (f.eks. tændt/slukket for en lampe eller en motor). En 'værdi'-aktuator (*value*) beskriver en aktuator, der kan antage mange forskellige værdier/niveauer, f.eks. lysstyrken på en lampe eller en temperatur.

Opbygningen af en generel aktuatorbeskrivelse er som følger:

- Yderst er property-grupperingen (med key='actuators'), der benyttes til at angive at alle efterfølgende properties beskriver eksterne aktuatorer.
- Hver enkelt sensor angives med en key, som benyttes internt i objektmodellen til at referere til objektet.
- Under de enkelte sensorer findes en række forskellige properties. Disse properties afhænger af hvilken type af aktuator det er, bortset fra to properties, der er fælles for dem begge.
    - *'type'* angiver hvilken type aktuator det er (state eller value), som beskrevet ovenfor.
    - 'name' angiver et aktuator id, der benyttes på den eksterne enhed. Dette id benyttes under kommunikationen med den eksterne enhed, til at få den rigtige aflæsning (se afsnit 4.3 *Kommunikationsprotokol* senere i dokumentet).
    - *'display'* er det navn, der skal vises i brugerfladen når værdien vises.
    - Værdiaktuator-properties
        - *'visualType'* angiver hvilken type visning der skal benyttes til at repræsentere værdien (se Tabel 3 herunder).
        - *'unit'* fortæller hvilken enhed denne sensoraflæsning er i.
          (denne property er ikke påkrævet)
        - *'min/max'* er eventuelle minimum- og maksimumværdier for aflæsningen.
    - Tilstandsaktuator-properties
        - 'states'-grupperingen indeholder alle de tilstande, der er mulige for aktuatoren

Værdiaktuatorer kan, som nævnt, være af forskellige typer, der angiver hvordan en ny aktuatorværdi kan vælges af brugeren.

Type	Eksempel
Simple Værdien vises som tekst i en kasse.	50°C
Slider Værdien vises som en linje hvor brugeren kan trække en markør til den rigtige værdi.	0 100

Tabel 3: Typer af værdiaktuatorer

Visning af tilstandsaktuatorer bestemmes af den platform, hvorpå historien afvikles (f.eks. af mobilapplikationen). F.eks. kan en slider vises som horisontal valuebar på platforme der ikke har sliders, eller som en slider med '+' og '-' knapper på platforme hvor en slider ikke umiddelbart vil kunne manipuleres i den grafiske brugerflade.

3.2.4 Diskussion af udvidelser af konfigurationsprotokollen

Den dynamiske grænsefladekomponent der er blevet tilføjet MOURDA-frameworket, benytter sig af en række primitiver til at bygge den grafiske brugerflade op. Disse primitiver er bundet til den funktionalitet man ønsker at de skal visualisere, f.eks. en aktuator eller en sensor (se afsnit 3.3.3 *Design af dynamisk komponent*).

Hvis systemet skal kunne håndtere mange generiske former for input og interaktion, vil det være nødvendigt at udvide mængden af tilgængelige primitiver. Da systemet ønskes at kunne håndtere næsten vilkårlige brugsscenarier betyder dette at mængden af primitiver bliver stor; der skal være en simpel valgkomponent, en mulighed for at foretage multiple valg, muligt at aflæse en række strenge osv. Men disse primitiver er implementeringsspecifikke og hører ikke til i en deskriptiv konfigurationsprotokol.

En idé til en udvidelse af protokollen kommer fra to af de relaterede projekter; *XForms* (2.1.7) og *TERESA* (2.1.9). I begge disse projekter er grænsefladebeskrivelserne mere abstrakte og beskrevet på et højere plan. I stedet for at beskrive ønsket implementering, beskriver de ønsket funktionalitet. Beskrivelserne benytter en række højniveau grafiske primitiver, f.eks. input og output, som blot angiver hvad den konkrete komponent skal kunne. Disse meget abstrakte primitiver gør det muligt at beskrive en brugergrænseflade uden at tag e hensyn til implementeringen.

I specialets konfigurationsprotokol skal der derfor benyttes mere generelle primitiver og enkelte højniveauprimitiver. Aktuator-klassen kan blot opfattes som et specialiseret input-primitiv, og sensor som et output-primitiv (set fra den eksterne enhed). Dette gælder for alle interaktionskomponenter der benyttes i den dynamiske komponent. Indførsel af et input- og et output-primitiv vil gøre det muligt for de eksterne enheder at specificere ønsket funktionalitet i brugergrænsefladen, frem for som nu at beskrive den ønskede implementering.

Implementeringen holdes derved væk fra beskrivelsen og protokollen vil gøre det muligt at beskrive meget mere komplicerede, og kraftfulde, interfaces. I stedet for at være nødt til at implementere en række ens inputkomponenter og kræve at XML-beskrivelsen skal vælge imellem dem, kan '*select*'-input f.eks. beskrive mange forskellige valgkomponenter, alt afhængig af de andre properties og den platform der benyttes. Valget af konkret interfacekomponent foretages så på runtime af mobilapplikationen.

Eksempel på en abstrakt beskrivelse

Ved at benytte højniveau grafiske primitiver bliver det muligt at udvide konfigurationsprotokollen uden at skulle tage hensyn til implementeringen. F.eks. kan man beskrive en sensorenhed ved at angive at det er outputkomponent med en værdi, i stedet for en sensor. Mobilapplikationen mapper så denne beskrivelse til en relevant outputkomponent, der enten kan være et tekstfelt, en valuebar eller andet (se Eksempel 18).

```
<property key='output'>
  <property key='value'>
    <property key='id'> light </property>
    <property key='name'>Lys</property>
    <property key='min'>0</property>
    <property key='max'>100</property>
    <property key='update'>
      <property key='type'> auto </property>
      <property key='interval'> 5 </property>
    </property>
  </property>
</property>
```

Eksempel 18: Generisk outputbeskrivelse

'`update`'-elementet i eksemplet er tilføjet protokollen for at give mulighed for at angive, hvor værdien af outputtet kommer fra.

Det samme er muligt med en aktuator, hvor der benyttes input i stedet for output. En tilstandsaktuator beskrives f.eks. som en 'select'-komponent med en række valgmuligheder. Mobilapplikationen kan så vælge at vise denne komponent med de grafiske primitiver den har til rådighed (f.eks. en række knapper).

```
<property key='input'>
  <property key='id'> pump </property>
  <property key='type'> select </property>
  <property key='name'> Bevaegelse </property>
  <property key='options'>
    <property key='1'>Start</property>
    <property key='2'>Stop</property>
  </property>
</property>
```

Eksempel 19: Generisk inputbeskrivelse

Udvidet interaktion

Udover tilføjelsen af mere generiske grafiske komponenter skal der også tilføjes en mulighed for at beskrive funktionaliteten i en dynamisk komponent mere generelt. I den nuværende implementering benyttes moduler og eventhandlers til at beskrive hvordan interaktionen påvirker historien og hvordan elementerne i brugergrænsefladen opfører sig. Men på samme måde som med de grafiske primitiver, er protokollen at man skal angive hvilken type af eventhandler man vil benytte og binde den til implementeringsspecifikke elementer.

Inspirationen til denne udvidelse kommer også fra *XForms* og *TERESA*-projekterne, hvor der, i *XForms*, benyttes 'triggers' til at beskrive hvad der skal ske når brugeren interagerer med brugerfladen og i *TERESA* benyttes abstrakte operatorer til at binde de forskellige grafiske elementer sammen.

I stedet for denne tætte kobling mellem funktionsbeskrivelse og implementering, skal der indføres generiske interaktionsbeskrivelser (triggers). F.eks. kan man angive at en bestemt handling skal afvikles når der interageres med en abstrakt interfacekomponent. Dette kan f.eks. være at skifte til en anden knude, opdatere en værdi eller vise/skjule et element. Ved at benytte mere generiske beskrivelser bliver det op til den aktuelle implementering at vælge hvordan interaktionen skal repræsenteres og afvikles.

Dette vil gøre det muligt at lave mere komplekse dynamiske komponenter uden at skulle binde beskrivelsen op på en specifik implementering. F.eks. er det ikke nødvendigt at en 'select'-komponent vises vha. en række knapper, og det er derfor ikke entydigt (i den nuværende form) hvordan man skal beskrive en handling der skal udføres når et valg træffes i interfacet.

Eksempel

Abstrakte interaktioner kan angives ved at beskrive hvad og hvornår der skal udføres, f.eks. kan der sættes en trigger på select-komponenten som opdaterer en værdi når et valg træffes:

```
<property key='input'>
  <property key='id'> pump </property>
  <property key='type'> select </property>
  <property key='name'> Bevaegelse </property>
  <property key='options'>
    <property key='1'>Start</property>
    <property key='2'>Stop</property>
  </property>
  <property key='triggers'>
    <property key='trigger1'>
      <property key='onSelect'> 1 </property>
      <property key='action'> update </property>
      <property key='target'> pump </propert>
    </property>
  </property>
</property>
```

Eksempel 20: Trigger på select-komponent

Triggeren kan så (i *MOURDA*) blive oversat til en ButtonPressedHandler, der bindes sammen med det ButtonModule som select-komponenten er implementeret ved. Eller hvis select-komponenten er implementeret vha. en anden grafisk komponent, bindes der et tilsvarende event op på den. Ideen er netop at beskrivelsen ikke skal tage hensyn til implementeringen.

Der kan også skiftes til en anden knude (f.eks. en alarmdialog) hvis et output antager en bestemt værdi:

```
<property key='output'>
  <property key='value'>
    <property key='id'> light </property>
    <property key='name'>Lys</property>
    <property key='min'>0</property>
    <property key='max'>100</property>
    <property key='update'>
      <property key='type'> auto </property>
      <property key='interval'> 5 </property>
    </property>
    <property key='triggers'>
      <property key='trigger2'>
        <property key='onValue'> 85 </property>
        <property key='action'> gotonode </property>
        <property key='target'> alertscreen </propert>
      </property>
    </property>
  </property>
</property>
```

Eksempel 21: Alert trigger på outputværdi

Hvor triggeren i dette tilfælde oversættes til en endnu ikke implementeret eventhandler, der aktiverer en GotoNodeAction når den fastsatte værdi opnås.

3.3 Implementering

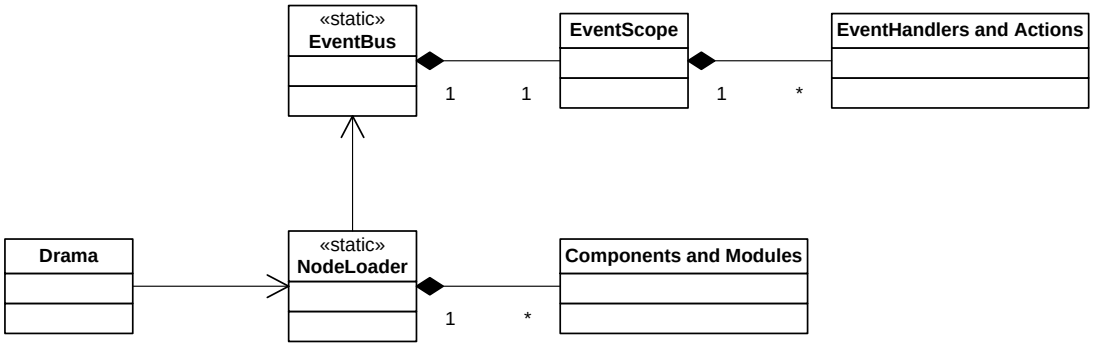
I dette afsnit beskrives implementering på mobilplatformen af de arkitekturændringer, der blev beskrevet i afsnit 3.1, og den konfigurationsprotokol, der blev beskrevet i afsnit 3.2.

3.3.1 Dynamisk udvidelse af historien

Tilføjelsen af det globale scope (se afsnit 3.1.4) har gjort det muligt løbende at tilføje elementer (historieknuder, eventhandlers og lignende) til en igangværende historie. Denne funktionalitet benyttes til at udvide historien dynamisk efterhånden som der findes nye eksterne enheder.

Globalt scope

I sin nuværende implementering har hvert drama et scope, der er implementeret i klassen EventScope, og hvor instansen er indeholdt i klassen EventBus (der er en statisk klasse, som refereres fra dramaet). I dette scope findes alle eventhandlers og deres actions. Samtidig indeholder den statiske klasse NodeLoader, de komponenter og moduler der er tilgængelige for det aktuelle drama.

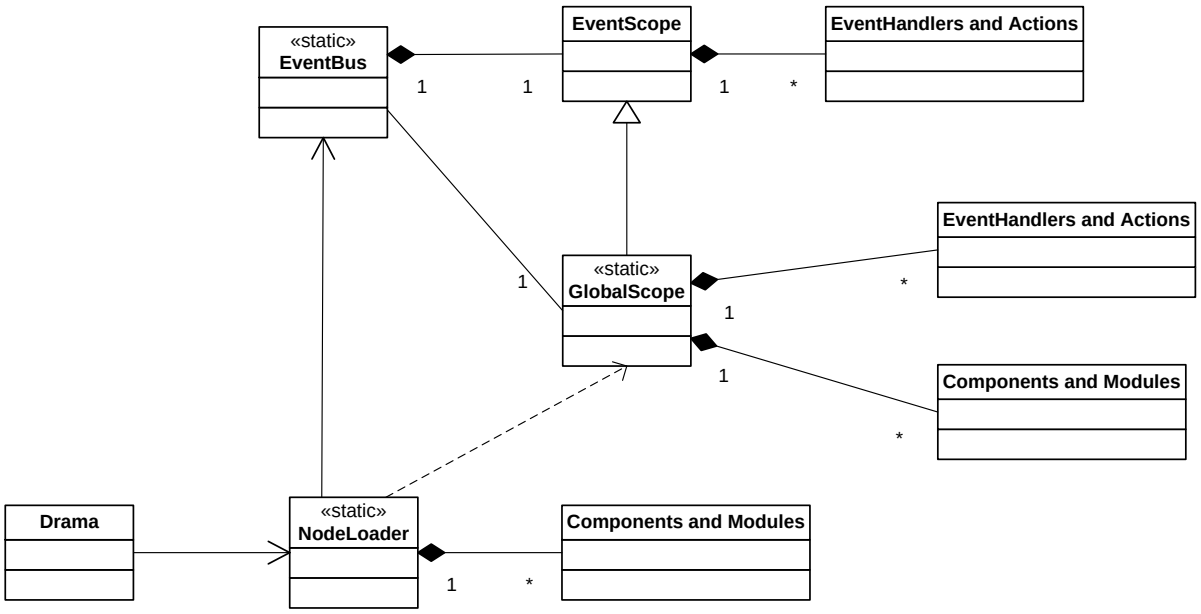


Figur 29: Eksisterende scope

Komponenterne i NodeLoaderen lever kun så længe der er et drama, der peger på dem. Hvis dramaet afsluttes, slettes disse elementer også. Samtidig sørger NodeLoaderen for, at EventScope-objektet skiftes ud, hver gang en ny knude bliver startet.

Det er derfor ikke muligt at have dynamiske komponenter og event handlers, der kan fungere korrekt uanset strukturen af, og flowet i, det underliggende drama, da de dynamiske elementer også vil blive nedlagt når de statiske elementer (dem der er beskrevet i dramaet) bliver nedlagt.

Det globale scope er blevet implementeret som en statisk udvidelse af EventScope-klassen, og denne er også indeholdt i EventBus-klassen. Ved at gøre den nye klasse statisk, sikres det at den ikke bliver nedlagt hver gang der skiftes mellem knuder. Samtidig er det muligt at tilføje komponenter og eventhandlers i ét scope og hente det ud i et andet.



Figur 30: Global scope

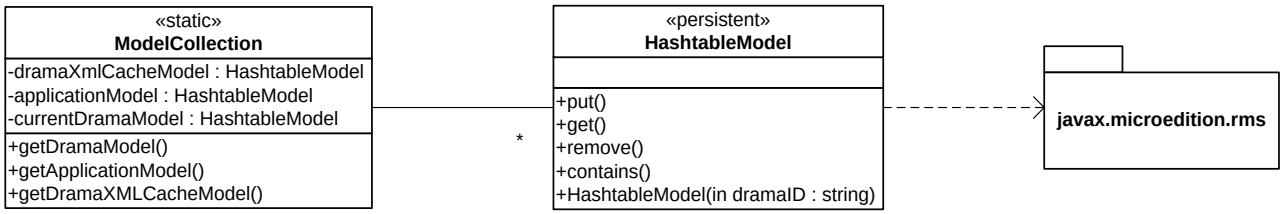
Som det kan ses i Figur 30 benytter NodeLoaderen sig af det nye globale scope. Her kan den hente de komponenter, moduler og event handlers der er blevet tilføjet (eller tilføje nye), uden at ændre på det igangværende drama.

NodeLoaderen kigger altid først i EventScope for at finde event handlers og først bagefter i det globale scope og ligeledes først i sin egen samling af komponenter før der ledes i det globale scope.

Persistent model

MOURDA-frameworket implementer en persistent model, der benyttes til at gemme data under programafviklingen. Modelhåndteringen er implementeret igennem en statisk klasse, ModelCollection, der indeholder tre forskellige persistente modeller; dramaXMLCacheModel, currentDramaModel og applicationModel.

Hver af disse modeller er implementeret som en HashtableModel. HashtableModel benytter Java ME RecordStoreManagement (RMS)[4] til at persistere data, der gemmes deri. Interfacet til denne klasse er lavet, så det er magen til interfacet for et almindeligt Java Hashtable.



Figur 31: Persistent model

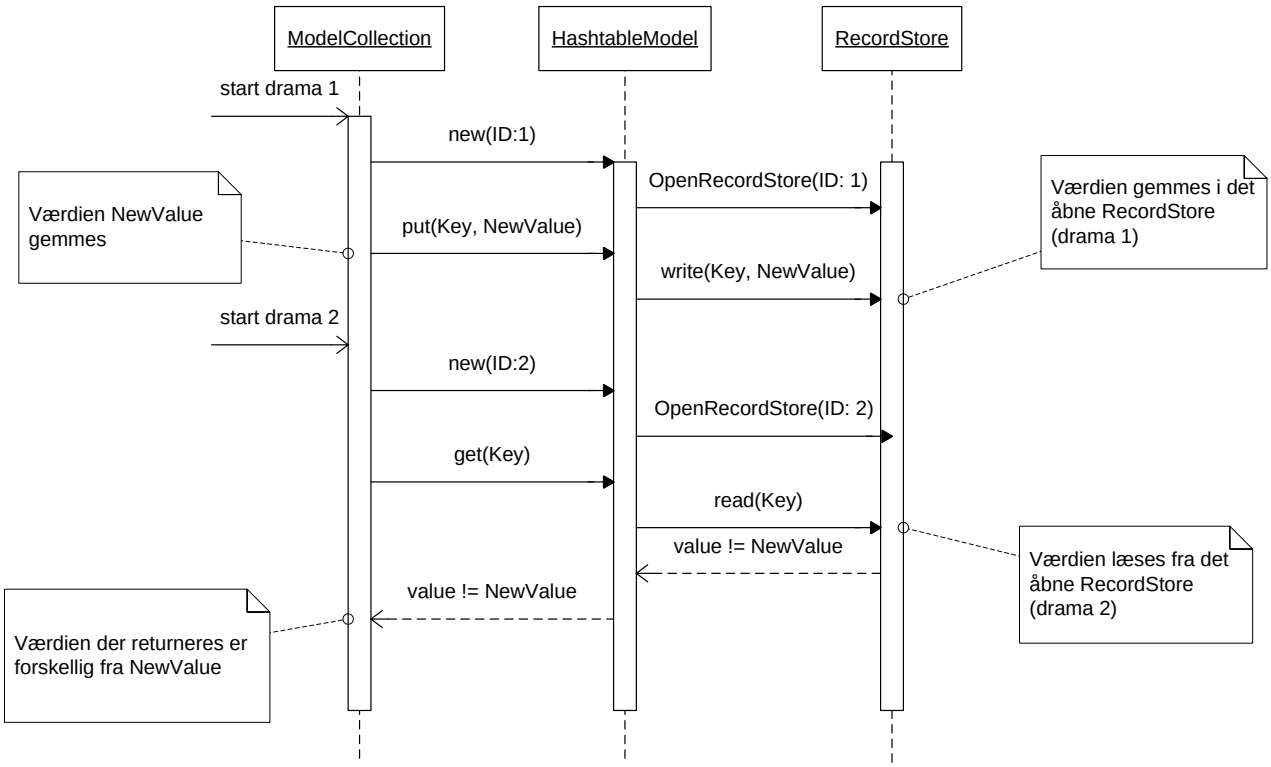
[4] http://developers.sun.com/mobility/midp/articles/databaserms/

DramaXMLCache-modellen indeholder de XML-beskrivelser, som applikationen har downloadet tidligere, forudsat at disse er dramabeskrivelser.

Dramamodellen er knyttet til et specifikt drama (modellen bindes til dramaets dramaid) og benyttes til at gemme data for det aktuelle drama, f.eks. hvor langt brugeren er kommet i historien eller hvilke valg der er truffet. Denne model indlæses når et drama startes og er ikke tilgængeligt for andre dramaer. Der oprettes en ny dramamodel for hver drama der startes.

Applikationsmodellen er global, så data gemt deri kan læses fra alle dramaer der måtte afvikles. Den benyttes til at gemme generelle informationer omkring brugeren af systemet, opsætninger og konfigurationsoplysninger.

Da det normalt er dramamodellerne der skal benyttes til at gemme data for igangværende dramaer, og da disse bliver lukket efter at et drama afsluttes, giver det problemer for de dynamiske knuder. Da de indlæses uafhængigt af det igangværende drama, kan de ikke benytte dramamodellen til at persistere data, da det ikke er sikkert at komponenten startes i samme drama næste gang (se Figur 32).



Figur 32: Dramamodellen skiftes i mellem dramaer

Derfor er de dynamiske knuder nødt til at benytte applikationsmodellen til at gemme data, for at være sikker på at det er tilgængeligt for komponenten næste gang denne startes, da denne model er global og kan tilgås fra alle dramaer.

Udvidelse af historien

Den dynamiske udvidelse af historien er derfor implementeret som funktionalitet, der tilføjer nye dynamiske knuder til det globale scope, hvorved de automatisk bliver tilgængelige for det igangværende drama (og også efterfølgende dramaer i samme programafvikling).

Ved at benytte sig af de eksisterende event-mekanismer, er det derfor blevet muligt at starte, stoppe og interagere med de nye komponenter, moduler og eventhandlers. Hvis et event peger på en ny komponent, vil det være muligt for NodeLoaderen at hente den frem og starte den, uden at ændre ved det eksisterende drama.

Ligeledes er det muligt at vende tilbage til det tidligere drama, fra en dynamisk knude, da dramaet ikke behøver at blive afsluttet og de eksisterende knuder stadig er tilgængelige i NodeLoaderen.

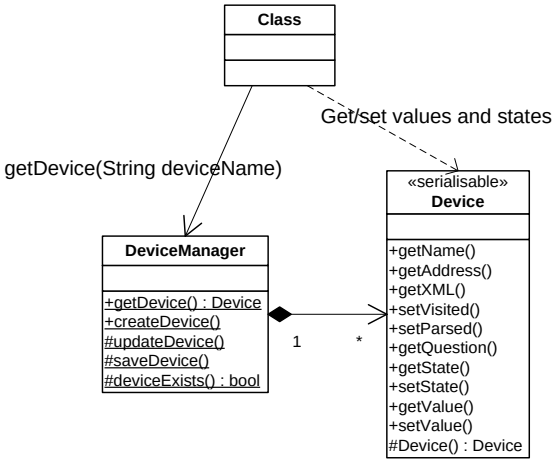
3.3.2 Moduler til kommunikation og discovery

Der er blevet implementeret en række klasser til at håndtere søgning efter, og kommunikation til, de eksterne Bluetooth-enheder.

Eksterne enheder

Alle eksterne enheder indkapsles i en Device-klasse, der repræsenterer de informationer som er tilgængelige om den aktuelle enhed. Klassen indeholder metoder til at hente XML, sætte og aflæse værdier (til sensorer/aktuatorer), hente det spørgsmål der stilles ved discovery og andre metoder der er nødvendige for at håndtere enheden korrekt. Device-klassen er serialiserbar så objekterne kan persisteres i RMS (Record Management System[5]) og kun behøver blive hentet én gang fra enheden.

Håndteringen af kendte devices er implementeret i en DeviceManager-klasse, hvor andre objekter, igennem en række statiske metoder, kan få fat på de devices der er brug for. Det er ikke muligt for objekter at instantiere devices direkte, de kan kun oprettes ved at kalde createDevice() i DeviceManager-klassen.



Figur 33: Device og DeviceManager

[5] http://developers.sun.com/mobility/midp/articles/databaserms/

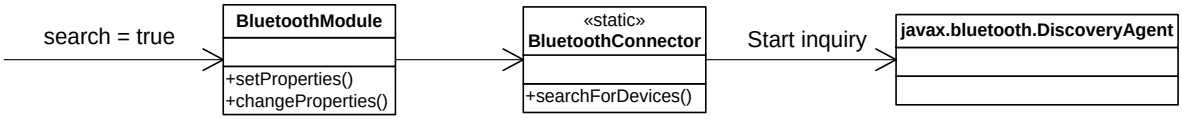
Modul og event

Der er blevet implementeret et modul, BluetoothModule, der er ansvarlig for søgningen efter eksterne enheder og håndteringen af de enheder der bliver fundet.

Modulet

Modulet er magen til de eksisterende moduler og implementerer det samme interface, men det har ingen synlige kommandoer. Modulet starter en Bluetooth-søgning når dets'search'-property sættes til 'true', enten ved at den sættes ved oprettelsen af modulet eller ved at værdien ændres vha. en aktion fra et event. Herefter starter en søgning efter nye enheder automatisk i baggrunden. 'Search' sættes automatisk til false efter endt søgning.

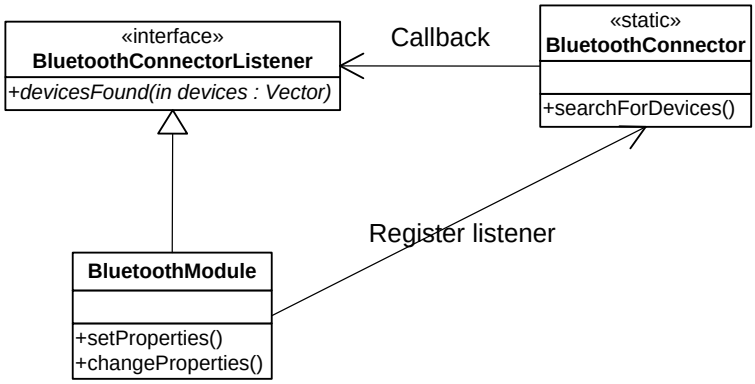
Den underliggende Bluetooth-håndtering er implementeret i en statisk klasse, BluetoothConnector, der starter en almindelig Java ME Bluetooth-søgning hvis der ikke allerede er en aktiv søgning i gang.



Figur 34: Start Bluetooth-søgning via modul

Interface

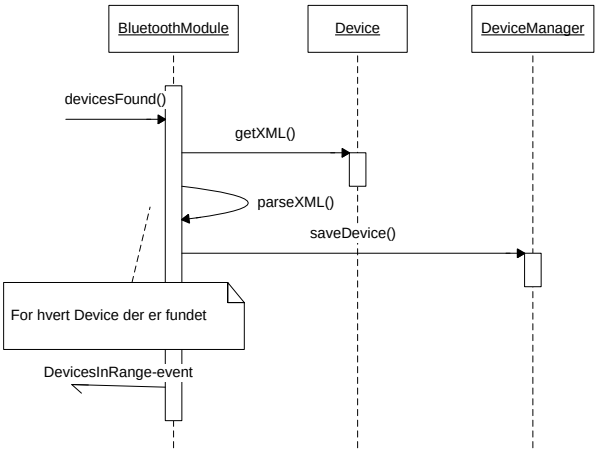
BluetoothModule-klassen implementerer samtidig et interface, BluetoothConnectorListener, som benyttes til at håndtere callbacks fra BluetoothConnector-klassen, når en søgning er afsluttet. Dette interface indeholder en enkelt metode, der som parameter tager imod en vektor med de fundne enheder.



Figur 35: Callback efter endt søgning

Events og actions

Modulet henter og indlæser automatisk XML for de nye enheder og gemmer nye enheder i DeviceManager-klassen (der er beskrevet i forrige underafsnit). Herefter sendes et DevicesInRangeHandler-event med navnene på de nye enheder som event-properties. Eventet håndteres igennem en global eventhandler.



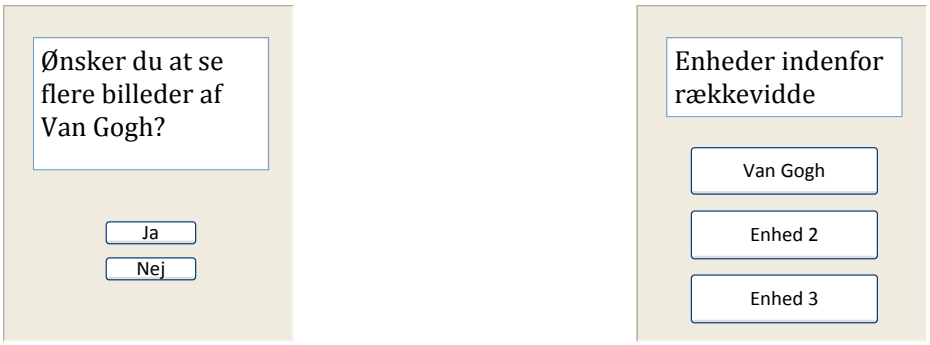
Figur 36: Sekvens for håndtering af fundne enheder

Der er implementeret en ny action til at håndtere visningen af fundne enheder; HandleDevicesAction. Denne action løber alle de fundne enheder igennem og kontrollerer om det er enheder, der allerede har været besøgt.

Hvis der findes en enhed der ikke er besøgt, vises en dialog der indeholder *discoveryQuestion* for enheden med det samme (Figur 37 venstre). Hvis brugeren svarer at hun gerne vil se enheden, sendes et OnLoadEvent med denne enheds navn (se delafsnittet *Nye konfigurationselementer* i afsnit 3.2.1 for en beskrivelse af discoveryQuestion og OnLoadEvent). Ellers fortsættes igennem listen af enheder.

Hvis der ikke findes nye enheder, eller brugeren har svaret nej til dem alle, vises enhederne i stedet som en liste over enheder indenfor rækkevidde (Figur 37 højre). Fra denne liste kan brugeren så vælge at starte en enhed, så længe den er indenfor rækkevidde, eller blot fortsætte med programmet. Listen vises af en grafisk komponent, DeviceChoice, der er implementeret som en standard MOURDA-komponent.

ID'et på denne komponent skal leveres til HandleDevicesAction-handleren som en property.



Figur 37: Visning af discoveryQuestion og liste over enheder

Hvis der ingen Bluetooth-enheder findes, sendes i stedet et BTSearchCompleteHandler-event, som indikerer at søgningen er afsluttet. En eventhandler kan så fange eventet og starte en ny søgning hvis dette er ønskeligt.

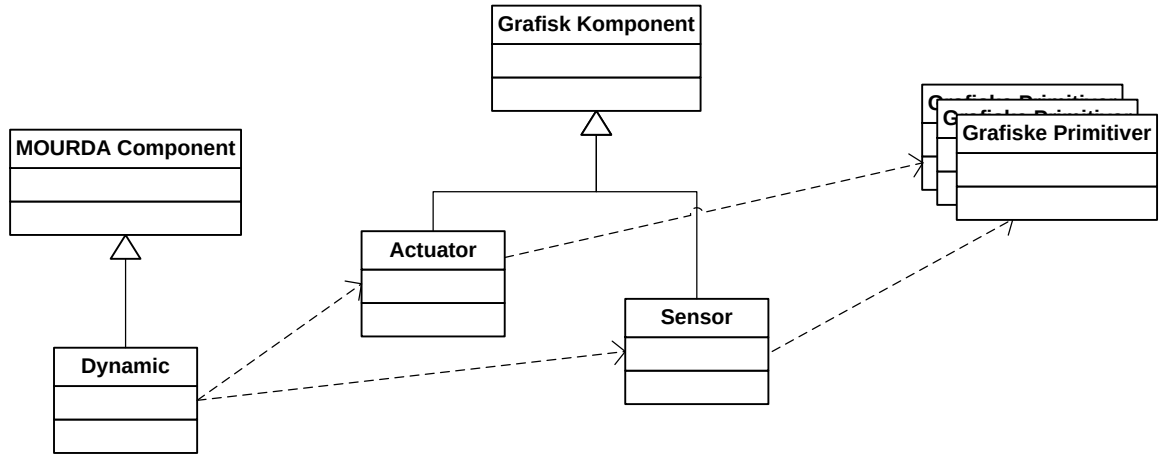
3.3.3 Design af dynamisk komponent

Der er blevet implementeret en ny MOURDA-komponent, Dynamic, til at håndtere visningen af de interaktive knuder der indeholder sensorer og aktuatorer. Klassen er opbygget på samme måde som de eksisterende komponenter i MOURDA, men den benytter to grafiske komponenter til at visualisere de to typer af enheder.

Sensor- og aktuatorelementerne i beskrivelsen (se afsnit 3.2.3) parses vha. to nye grafiske komponenter (klasserne Sensor og Actuator) der er beskrevet i afsnit 3.3.4. Brugen af disse grafiske komponenter mappes direkte til XML-beskrivelsen og håndteres automatisk i forhold til den dynamiske komponents beskrivelse, for at sikre at en vilkårlig (gyldig) beskrivelse kan mappes til en grafisk visualisering.

Dynamic-objektet gennemløber listen af sensor- og aktuatorelementer i XML'en og opretter en instans af den respektive komponent for hver indgang i listen (se Figur 38).

De ekstra properties (se afsnit 3.2.1), der er beskrevet i beskrivelsen, håndteres af Dynamic-komponenten ved at vise titel, tekst og style som en almindelig MOURDA-komponent.



Figur 38: Den dynamiske komponents afhængigheder

3.3.4 Grafisk visning af sensor- og aktuatorinformation

De to grafiske komponenter, Actuator og Sensor, repræsenterer den information, der er beskrevet i XML-beskrivelsen. Implementeringen af de forskellige muligheder i protokollen beskrives herunder.

Komponenterne er implementeret som Containers med grafiske primitiver. Primitiverne benyttes til at vise tekst, grafik, sliders, value bars og lignende, alt afhængig af typen og indholdet af elementet. Samtidig indkapsles de i en ramme, der visuelt afgrænser komponenten, så eventuelle ens valgmuligheder mellem komponenter ikke giver anledning til forvirring for brugeren.

Sensor og Actuator-klasserne implementerer den konfigurationsprotokol, der er beskrevet i afsnit 3.2.3. Brugen af primitiverne i Actuator og Sensor-klasserne er mappet direkte til XML-beskrivelsen, på samme måde som brugen af Actuator og Sensor-objekter er i Dynamic-klassen, hvilket sikrer at visningen af komponenterne opfylder beskrivelsen i den dynamiske komponent.

Aktuator

Der er to hovedtyper af aktuatorer; tilstand og værdi (state og value). Hver af disse hovedtyper er implementeret, så de håndterer de muligheder konfigurationsprotokollen giver.

Tilstand

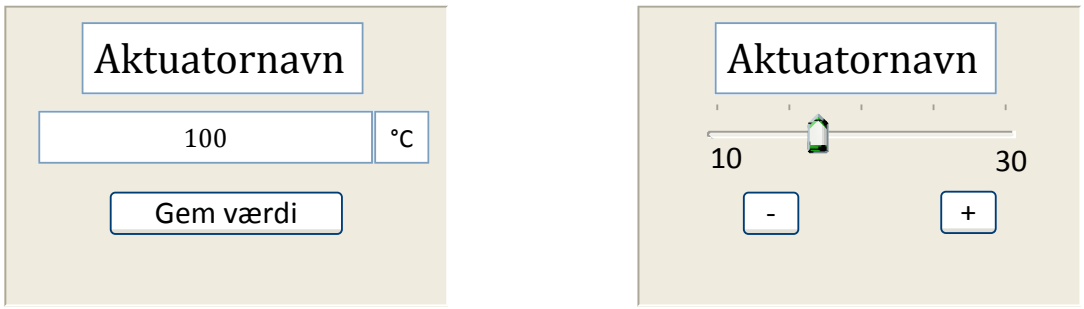
Tilstandsaktuatorer repræsenteres ved at aktuatorens navn og nuværende tilstand vises som tekststrenge.

De muligheder der er for at vælge tilstand, repræsenteres vha. grafiske knapper, hvorpå tilstandens navn skrives. Knapperne giver brugeren mulighed for at vælge en ny tilstand ved at benytte sig af telefonenss knap/joystick-interface til at vælge og aktivere en mulighed.

Figur 39: Tilstandsaktuator

Værdi

Værdiaktuatorer er implementeret ved at vise brugeren enten et inputfelt (hvis den visuelle type er simpel) eller en skyder/"slider" (hvis den visuelle type er slider). Inputfeltet har en knap tilknyttet, der giver brugeren mulighed for at gemme den aktuelle værdi. Skyderen styres vha. to knapper, der ændrer skyderens værdi i den ene eller den anden retning. Figur 40 viser de to muligheder for værdiaktuatorer, med 'simpel' til venstre og 'slider' til højre.



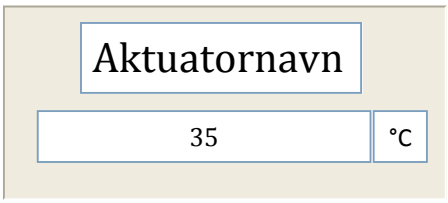
Figur 40: Værdiaktuatorer

Sensor

Sensorkomponenten håndterer de tre muligheder der er for typer af sensorer; simpel, horisontal og vertikal.

Simpel

Den simple sensorvisning består af to tekstfelter; en med sensornavn og en der viser den aflæste værdi fra sensoren.

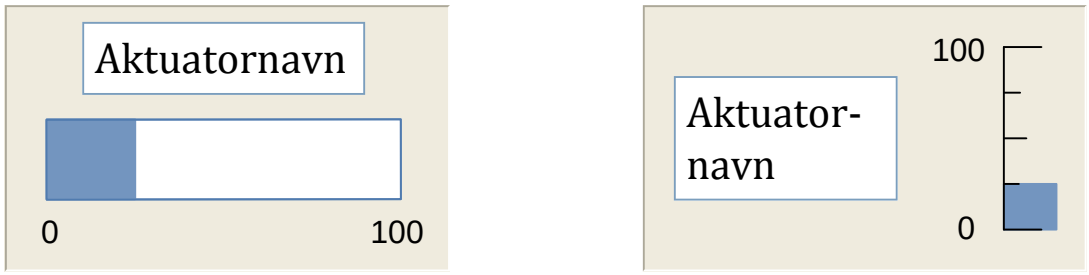


Figur 41: Simpel sensor

Horisontal og vertikal

Disse to typer er implementeret på den samme måde. Sensorværdien vises med et felt der bliver udfyldt procentvis i forhold til minimum og maksimum. Forskellen er hvilken vej feltet vender.

Feltet optegnes manuelt ved at udregne den procentuelle værdi og så tegne to udfyldte felter ovenpå hinanden med længder der svarer til det forhold der skal vises. Figur 42 viser eksempler på en horisontal (til venstre) og en vertikal (til højre) sensorvisning.



Figur 42: Sensorer med value bars

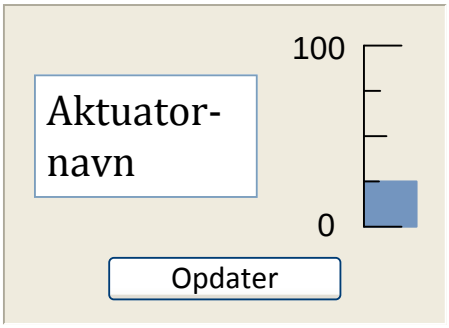
Opdatering af værdi

Håndteringen af værdiopdatering (se afsnit 3.2.3) er fælles for alle sensortyperne.

Hvis værdien ikke kan opdateres foretages der intet ekstra i forhold til eksempler der er vist ovenfor.

Hvis værdien opdateres automatisk, tilføjes der en Timer til objektet, som sørger for at hente en ny værdi ud fra den eksterne enhed, i det angivne interval. Timeren kører i baggrunden og er ikke synlig for brugeren

Hvis værdien skal opdateres manuelt, tilføjes der en knap nederst i komponenten, der giver brugeren mulighed for manuelt at opdatere værdien (se Figur 43 nedenfor).



Figur 43: Vertikal sensorkomponent med opdateringsknap

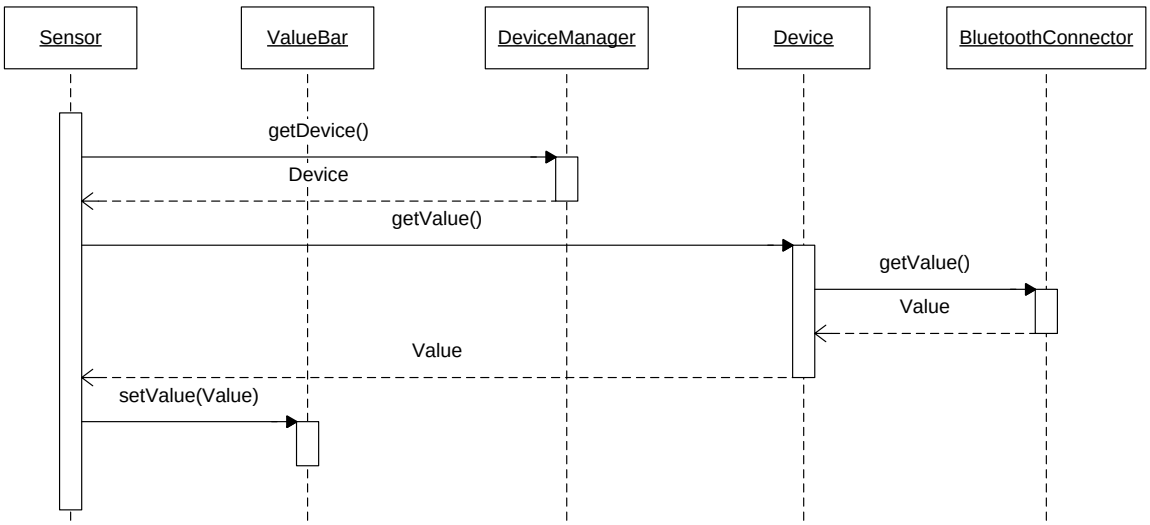
Værdier fra eksterne enheder

Hver sensorkomponent aflæser den aktuelle værdi for den sensor den svarer til ift. beskrivelsen. Actuator-komponenten aflæser ligeledes en værdi og har samtidig mulighed for at sætte en værdi på den eksterne enhed, når en bruger ændrer en værdi i interfacet.

Dette gøres ved at hente det aktuelle Device ud fra DeviceManager-klassen og benytte det interface der er til rådighed her, ved at kalde getValue/setValue og getState/setState-metoderne.

Håndteringen af den underliggende Bluetooth-kommunikation, der er krævet for at aflæse/sætte værdierne, er afkoblet fra den grafiske visning og implementeret i de tidligere nævnte klasser, som beskrevet i afsnit 3.3.2.

Visningen af sensorværdierne og repræsentationen af den mulige brugerinteraktion (ændring af tilstand eller værdi) er implementeret i de grafiske primitiver der er beskrevet i forrige delafsnit.



Figur 44: Aflæsning og visning af værdi

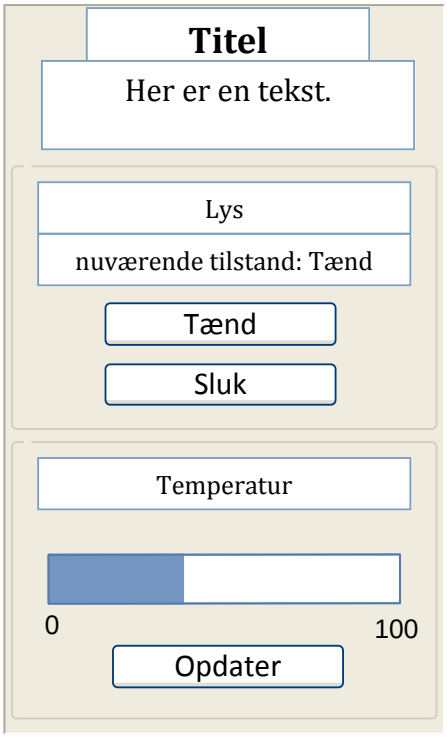
3.3.5 Eksempel på indlæsning af XML-beskrivelse

Følgende eksempelbeskrivelse viser hvordan der mappes fra XML til grafik. XML-beskrivelsen følger den protokol der er beskrevet i afsnit 3.2 (style-elementet er ikke medtaget).

```
<node id='node1' component='net.interactivespaces.mud.components.Dynamic'>
<properties>
    <property key='deviceName'>DeviceName</property>
    <property key='title'>Titel</property>
    <property key='textKey'> Her er en tekst. </property>
    <property key='actuators'>
        <property key='actuatorX'>
            <property key='type'> state </property>
            <property key='name'> light </property>
            <property key='display'> Lys </property>
            <property key='states'>
                <property key='1'>Tænd</property>
                <property key='2'>Sluk</property>
            </property>
        </property>
    </property>
    <property key="sensors">
        <property key="reading2">
            <property key="name"> Temp1 </property>
            <property key="display"> Temperatur </property>
            <property key="type"> Horizontal </property>
            <property key="update"> manual </property>
            <property key="unit"> C </property>
            <property key="max"> 100 </property>
            <property key="min"> 0 </property>
        </property>
    </property>
</properties>
</node>
```

Eksempel 22: Eksempel på XML-beskrivelse

De grafiske komponenter, der er beskrevet i de forrige delafsnit, indlæser denne beskrivelse og mapper den til en grafisk repræsentation, der ser ud som i Figur 45 (her antages det at værdierne fra enheden er aflæst til at være 'state = 1' for aktuatoren og 35[°C] for sensoren).



Figur 45: Eksempel på grafisk visning

Som det kan ses i eksemplet er alle elementer fra XML-beskrivelsen en del af den grafiske repræsentation og de muligheder der præsenteres for brugeren matcher de muligheder, som den eksterne enhed stiller til rådighed.

Hvad der ikke er umiddelbart synligt i eksemplet er den interaktion som brugergrænsefladen også repræsenterer. Knapperne i midten giver mulighed for at tænde og slukke for det lys som den eksterne enhed er ansvarlig for. Hvis en af mulighederne aktiveres vil den 'nuværende tilstand' automatisk blive opdateret til den aktuelle tilstand for lyset.

Flere eksempler på implementering af XML-beskrivelser kan ses i appendikset afsnit 9.2.1.

3.4 Opsummering

Konfigurationsprotokollen (beskrevet i afsnit 3.2) er blevet implementeret på mobilplatformen så den besvarer følgende delspørgsmål fra specialets formål:

Delspørgsmål I - *Hvordan repræsenteres indlejrede sensorer og aktuatorer grafisk på en mobiltelefon?*

Delspørgsmål II - *Hvordan skabes denne grafiske repræsentation dynamisk, uden at de indlejrede enheder og mobiltelefonen kender hinanden på forhånd?*

Disse spørgsmål er blevet besvaret igennem implementeringen af følgende elementer.

Den grafiske del af systemet, der afvikles på mobiltelefonen, er implementeret som en udvidelse af MOURDA-frameworket, som beskrevet i afsnit 1.3 (*Afgrænsning*). Udvidelsen bygger videre på eksisterende mekanismer i frameworket og bryder ikke med de design- og arkitekturprincipper der benyttes deri. Frameworkudvidelsen følger de retningslinjer der blev opstillet i afsnit 1.4.2 (*Baggrund for frameworket*) for konvertering af en XML-beskrivelse til en grafisk repræsentation.

Samtidig opfylder implementeringen de krav der blev opstillet i afsnit 1.5.1 (*Metode*) for en dynamisk konfiguration af brugergrænsefladen og en grafisk repræsentation af eksterne sensorer og aktuatorer:

(fra afsnit 1.5.1) *Herudover er formålet at få udvidet selve MOURDA-platformen, så den kan konfigureres og udvides dynamisk, ved at modtage nye beskrivelser fra eksterne enheder.*

Konfigurationsprotokollen implementeres ved at lave en modificeret version af MOURDA-platformens XML-beskrivelse af grafik og historie.

{…} Der skal tilføjes elementer til XML-beskrivelsen, der kan benyttes til at beskrive kommunikation med eksterne enheder, indsættelse og afvikling af nye historieelementer samt konfiguration af sensorstyring.

Dette opfylder følgende af specialets forventede resultater (se afsnit 1.6):

A3 (*Et interface til at aflæse og styre sensorer/aktuatorer, der er koblet til de indlejrede enheder*).

A4 (*Et interface til dynamisk at udvide en eksisterende historie, på mobiltelefonen, ved hjælp af en beskrivelse hentet fra en indlejret enhed*).

Kapitel 4. Indlejrede enheder og kommunikationsprotokol

I dette kapitel undersøges og beskrives de indlejrede enheder der benyttes i specialet og den kommunikationsprotokol der benyttes til at kommunikere med dem.

Først implementeres konfigurationsprotokollen, fra afsnit 3.2 *Konfigurationsprotokol*, på Arduino-platformen, så de indlejrede enheder kan opbygge en XML-beskrivelse af deres funktionalitet.

Herefter analyseres og designes den kommunikationsprotokol, der skal benyttes til kommunikation i mellem mobilapplikationen og de indlejrede enheder. Herunder hvorledes historien overføres, samt hvordan sensorer/aktuatorer aflæses og styres.

Formålet med analysen er at sikre at implementeringen opfylder to af delspørgsmålene fra specialets formål (afsnit 1.2):

Delspørgsmål III (*Hvordan håndteres fysisk interaktion med indlejrede enheder fra en mobiltelefon?*).

Delspørgsmål IV (*Hvordan skabes den fysiske interaktion igennem et interface til indlejrede enheder?*).

Til sidst analyseres og designes den service discovery protokol, der benyttes i systemet til at give mobilapplikationen mulighed for at finde de eksterne enheder.

4.1 Undersøgelse af Arduino-platformen

Arduino-platformen er, som tidligere nævnt (i afsnit 1.3 *Afgrænsning*), en udbredt prototypeplatform, der samtidig er fuldstændig open source.

Platformen giver en base til at udvikle prototypen til dette speciale, der ikke har karakter af at være en prototype, men som derimod benytter sig af det samme hardware og de samme softwareteknikker som en endelig udgave.

4.1.1 Hardware

Platformens hardware er også open-source og skemaerne, der beskriver elektronikken, er frit tilgængelige, hvilket giver mulighed for at udvide platformen som man ønsker.

Samtidig er det muligt at købe færdige reference-boards og der findes et væld af sensorer, aktuatorer og udvidelsesmoduler til platformen, så man kan tilpasse hardwaren til de behov der eksisterer i et givent projekt. Alle enhederne har en række analoge og digitale I/O-porte som frit kan benyttes til at tilslutte eksterne sensorer, aktuatorer og andet hardware med.

Selve controllerhardwaren er baseret på Atmel ATmega microcontrolleren, der er en udbredt og velkendt controller serie. I dette speciale benyttes ArduinoBT hardwaren med en ATmega168 controller.

Hardwaren stiller en række begrænsninger til programmet, og mere specifikt til den kommunikationsprotokol der skal udvikles, da ATmega168 kun har 16Kb flashhukommelse og kun 1Kb ram til programafvikling [6]. Denne hukommelse skal, udover protokolhåndteringen, også indeholde den XML-beskrivelse som enheden skal sende til mobiltelefonen. Clockhastigheden på ATmega168 microcontrolleren er 20 MHz, hvilket skal tages med i betragtning når protokollen implementeres.

Konfigurationsprotokollen kan ikke ændres ret meget da den er afledt af MOURDA's generelle interfacebeskrivelse, hvorfor kommunikationsprotokollen skal være simpel og effektiv for at overholde de hardwarekrav platformen stiller.

Bluetooth-hardwaren på ArduinoBT er en Bluegiga WT11 enhed[7], der er integreret med Arduino-hardwaren igennem en emuleret serielforbindelse.

Enheden er klasse 1 og understøtter Bluetooth 2.1+ EDR, dog er der igennem serielinterfacet kun mulighed for en forbindelseshastighed på 115,2 Kb/s.

4.1.2 Software

Arduino platformen benytter et udviklingssprog, der er baseret på C/C++ [8]. De fleste almindelige funktioner og typer fra C er tilgængelige og kodestrukturen er letforståelig for personer med erfaring indenfor disse sprog.

Derudover stilles der en lang række biblioteker til rådighed, som kan benyttes til at kommunikere med enten tilsluttet hardware eller eksterne enheder.

Selve programkoden er inddelt i 3 hovedelementer; setup, loop og andre funktioner.

- Det første der kaldes under programafviklingen er funktionen setup(), der er ansvarlig for at initialisere hardwaren og eventuelle variable i koden.
- Herefter kaldes funktionen loop() kontinuerligt indtil enheden bliver genstartet. Det er her selve programlogikken udføres.
- De andre funktioner kan så kaldes fra loop()-metoden for at håndtere den funktionalitet som programmet stiller til rådighed.

Dette giver alt sammen mulighed for at implementere en simpel og overskuelig protokol, der samtidig er effektiv så den kan afvikles på Arduino-hardwaren.

[6] http://www.atmel.com/dyn/products/Product_card.asp?part_id=3303
[7] http://www.bluegiga.com/WT11_Class_1_Bluetooth_Module
[8] http://arduino.cc/en/Reference/Extended

4.2 Konfigurationsprotokol på Arduino

Konfigurationsprotokollen benyttes indirekte på de eksterne Arduino-enheder, til at opbygge en beskrivelse af deres funktionalitet.

XML-beskrivelsen genereres før programkoden overføres til Arduino-enheden. Beskrivelsen laves så den matcher den funktionalitet som enheden stiller til rådighed, ved at beskrive de sensorer og aktuatorer den har eller ved at beskrive den historieudvidelse som enheden er ansvarlig for.

XML'en følger den protokol, der er beskrevet i afsnit 3.2, så programmet på mobiltelefonen kan indlæse og vise de knuder der er beskrevet deri. Navnet på enheden, sensorer og aktuatorer i beskrivelsen, matcher de navne der er implementeret på Arduino-enheden, jævnfør kommunikationsprotokollen (se afsnit 4.3)

XML-beskrivelsen gemmes som en tekststreng (eller hvis enheden tillader det, som en fil), vis indhold sendes til mobiltelefonen når denne spørger efter den.

4.3 Kommunikationsprotokol

Kommunikationen mellem mobilapplikation og de eksterne enheder benytter Bluetooth-protokollen som underliggende fysisk kommunikationslag. Oven på denne protokol benyttes en kommunikationsprotokol, der er specifik for denne udvidelse af MOURDA-frameworket.

Kommunikationsprotokollen gør det muligt for mobilapplikationen at hente XML-beskrivelsen af den eksterne enhed, samt at aflæse sensorer eller påvirke aktuatorer.

4.3.1 Analyse og design af protokollen

Formålet med protokollen er at skabe en mulighed for at aflæse og skrive informationer fra og til den eksterne enhed. Samtidig skal protokollen være simpel, for at sikre at den kan implementeres og afvikles på de meget begrænsede Arduino-enheder (se afsnit 4.1.1 for en beskrivelse af hardwaren).

Protokollen baseres på overførsel af beskrivende strenge, der er termineret med en bestemt karakter. Dette gøres for at holde protokollen simpel og effektiv ift. de muligheder som Arduino-enhederne har til rådighed. Termineringskarakteren er valgt til at være ';' (semikolon).

Samtidig benyttes der en separeringsstreng, ':' (kolon), til at adskille parametre fra hinanden i en overførsel.

Dette medfører samtidig at karaktererne ';' og ':' ikke er tilladt i sensornavne, værdier eller tilstande.

Det er muligt at udføre følgende tre handlinger via denne protokol: Hente XML, aflæse en værdi og sætte en værdi.

Forbindelse

Oprettelse af forbindelsen imellem Arduino-enheden og mobilen er ikke en del af protokolbeskrivelsen, da denne antages at være håndteret korrekt af de underliggende systemer.

Alle informationer sendes via Arduino-enhedens serielle interface, med den højeste Baud-rate der er tilgængelig.

Protokollen stiller ingen krav til hvordan information aflæses eller sendes på mobiltelefonen, dog skal det være kompatibelt med den underliggende serielle Bluetooth-datastrøm.

Overførsel af XML-beskrivelse

For at modtage en XML-beskrivelse skal mobilapplikationen sende følgende streng til den eksterne enhed:

```
XML;
```

Strengen 'XML' efterfulgt af termineringskarakteren semikolon som vist.

Dette medfører at navnet 'XML' er et reserveret navn og derfor ikke må benyttes som sensor-id i XML-beskrivelsen.

Herefter sender Arduino-enheden længden af XML-beskrivelsen (i antal bytes) efterfulgt af hele XML-beskrivelse, til mobilapplikationen, der så kan parse den. Beskrivelsen sendes derfor på formattet:

```
LÆNGDE:XML-BESKRIVELSE
```

Kommunikation

Selve den interaktive kommunikation (aflæsning/skrivning af værdier og tilstande) håndteres ligeledes vha. en simpel protokol.

Aflæsning af sensorer

For at aflæse en sensorværdi, skal mobilapplikationen sende følgende streng:

```
SensorID;
```

Sensorens id som en streng efterfulgt af termineringskarakteren semikolon, som vist. Hvis sensoren f.eks. har id'et Temp1, sendes '`Temp1;`'

Herefter sender Arduino-enheden værdien for den valgte sensor som en streng efterfulgt at termineringskarakteren. F.eks.: `23;`

Det samme gælder for aflæsning af en aktuatorværdi.

Styring af aktuatorer

For at sætte en aktuatorværdi, skal mobilapplikationen sende følgende streng:

```
AktuatorID:Værdi;
```

Aktuatorens id som en streng efterfulgt af separeringsstrengen kolon, efterfulgt at den nye aktuatorværdi/-tilstand efterfulgt af termineringskarakteren.

Hvis man f.eks. ønsker at sætte tilstanden 'Sluk' (tilstand '2') for aktuatoren 'Lys' sendes '`Lys:2;`'.

Herefter returnerer den eksterne enhed den nye værdi/tilstand for aktuatoren, på samme måde som værdien returneres efter en aflæsning af en sensor. F.eks. returneres '`2;`' efter forrige kald, forudsat at dette er den nye tilstand for lyset.

Dette gøres for at mindske antallet af kald til den eksterne enhed, da det ellers ville være nødvendigt for mobilapplikationen at foretage et nyt kald for at få den nye værdi/tilstand for aktuatoren at vide.

Fejlhåndtering i protokollen

Da en af protokollens hovedformål er at være simpel, er fejlhåndteringen deri også simpel.

Hvis der sendes en ugyldig forespørgsel, der mangler en parameter eller der forsøges at sætte en ugyldig værdi, returneres strengen '`Error;`' fra Arduino-enheden.

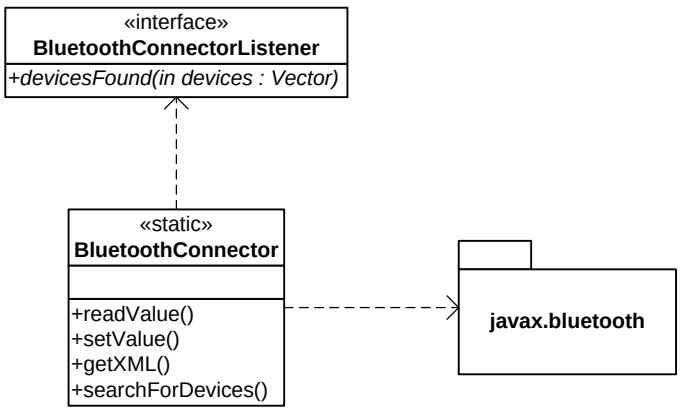
Det er så op til den kaldende enhed (mobilapplikationen) at håndtere denne fejl og forsøge igen.

4.3.2 Implementering i mobilapplikationen

Protokolhåndteringen i mobilapplikationen er implementeret i to klasser; BluetoothConnector og BluetoothDeviceConnection.

BluetoothConnector

BluetoothConnector er en statisk klasse, der agerer interface imellem den fysiske Bluetooth-forbindelse og resten af applikationen. Denne klasse står for at søge efter, og forbinde til, de eksterne enheder vha. de underliggende Java ME kommunikationsklasser i javax.bluetooth pakken. Klassen er kort beskrevet i afsnit 3.3.2 (*Moduler til kommunikation og discovery*) og implementeringen vil blive uddybet her.



Figur 46: BluetoothConnector og afhængigheder

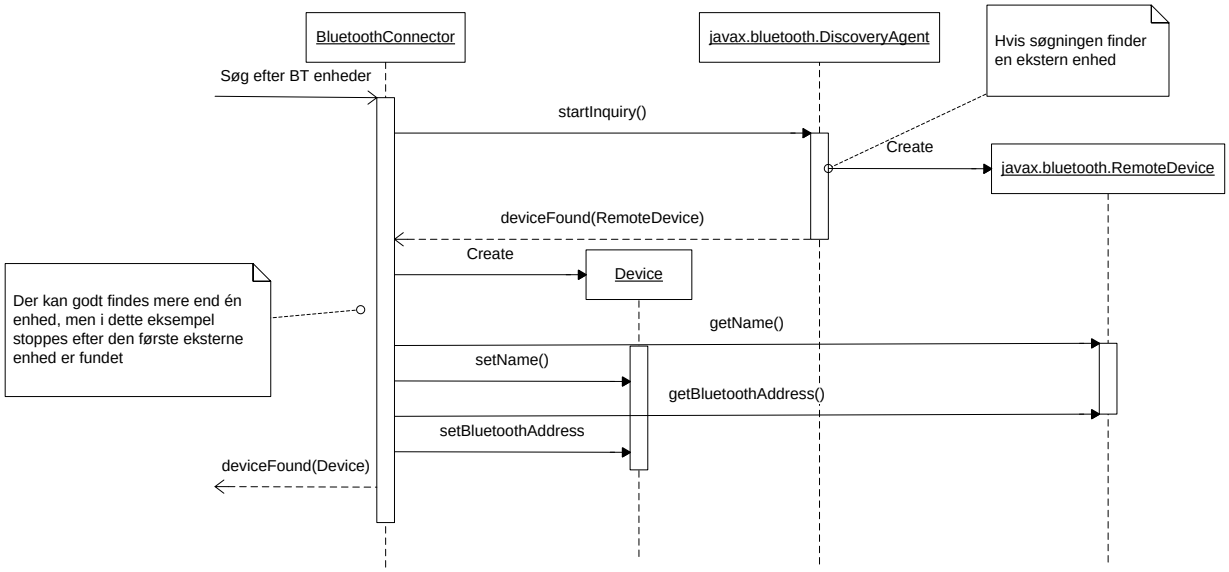
Søgning

Klassen stiller metoder til rådighed som gør det muligt for applikationen at starte en søgning efter eksterne enheder og, igennem et interface (BluetoothConnectorListener), at få svar tilbage når enheder er blevet fundet. Der startes kun en ny søgning, hvis der ikke allerede er en søgning i gang.

Denne funktionalitet er implementeret vha. javax.bluetooth.DiscoveryAgent-klassen der håndterer den underliggende Bluetooth-søgning efter enheder. Derfor implementerer BluetoothConnector-klassen

også javax.bluetooth.DiscoveryListener-interfacet der benyttes til at få svar fra DiscoveryAgent når Bluetooth-enheder er fundet.

Hver gang en enhed bliver fundet, pakkes denne ind i et Device-objekt (se delafsnit *Eksterne enheder* i afsnit 3.3.2) og tilføjes en vektor. Hvis enheden ikke tidligere har været vist, afsluttes søgningen. Efter endt søgning kaldes tilbage på BluetoothConnectorListener-interfacet med vektoren med de fundne enheder som argument.



Figur 47: Ekstern Bluetooth-enhed fundet ved søgning

Kommunikation

Samtidig stiller BluetoothConnector-klassen også metoder til rådighed, der gør det muligt at hente XML, aflæse værdier og sætte værdier på en ekstern enhed.

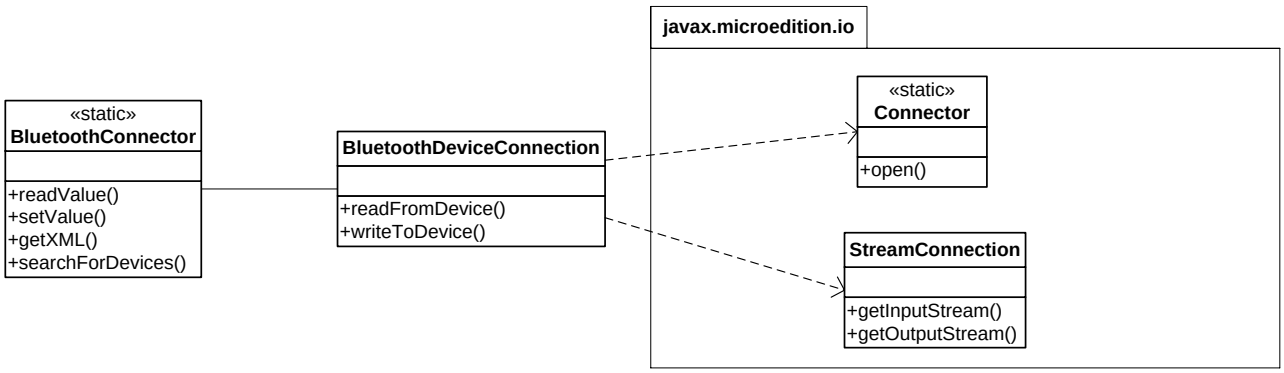
Disse metoder benytter sig af klassen BluetoothDeviceConnection til at håndtere kommunikationen med Arduino-enheden (se nedenfor).

BluetoothDeviceConnection

Denne klasse implementerer selve kommunikationsdelen med de eksterne enheder. Klassen stiller metoder til rådighed, til at læse fra og skrive til en Bluetooth-enhed (indkapslet i et Device-objekt).

Kommunikationen håndteres igennem en javax.microedition.io.StreamConnection og forbindelsen til den eksterne enhed oprettes igennem statiske metoder på klassen javax.microedition.io.Connector. Forbindelsen oprettes via BTSPP-protokollen (Bluetooth Serial Port Profile[9]), der emulerer en standard RS-232 seriel forbindelse. Denne profil benyttes for at understøtte den underliggende serielle kommunikation som Arduino-enhederne benytter (se delafsnittet *Forbindelse* i afsnit 4.3.1).

[9] http://www.bluetooth.com/English/Technology/Works/Pages/SPP.aspx



Figur 48: BluetoothDeviceConnection og afhængigheder

Forbindelse til den eksterne enhed holdes åben så længe enheden er aktiv i mobilapplikationen. Dette gøres for at sikre en høj hastighed ved aflæsning/ændring af værdier og tilstande på enheden. Dog betyder dette at den eksterne enhed muligvis bliver blokeret for andre instanser af mobilapplikationen, da kommunikationsprotokollen ikke kan håndtere multiple klienter.

Det er også muligt at implementere kommunikationen, så forbindelsen lukkes ned efter hver aflæsning/ændring for at sikre imod at forbindelsen blokerer, men dette er ikke implementeret. Se afsnit 6.1.5 for en mere dybdegående evaluering af denne problemstilling, samt målinger og resultater fra undersøgelsen af den.

4.3.3 Implementering på Arduino-platformen

Protokolimplementeringen på Arduino-enhederne er, på samme måde som selve protokollen, holdt simpel for at sikre at den kan afvikles på alle Arduino-enhederne, samt at den er overskuelig og hurtig.

Kommunikation

Kommunikationen mellem Arduino-enheden og mobilapplikationen, er implementeret vha. Arduino-pakken Serial[10]. Denne pakke indeholder metoder til at læse og skrive fra/til en seriel port, som på Arduino-enhederne er den forbindelse der er til den fysiske Bluetooth-enhed.

Under opstart af boardet initieres den serielle port til den højest tilladte Baud-rate, ved at kalde Serial.begin(115200). Herefter er den serielle port klar.

Aflæsning fra porten foregår én karakter af gangen ved at kalde Serial.read() i en uendelig løkke.

Når der sendes svar tilbage til en Bluetooth-klient, benyttes metoden Serial.print(), der skriver en streng som karakterer til den serielle port.

[10] http://arduino.cc/en/Reference/Serial

Protokol

Protokollen er implementeret ved at der læses fra den serielle port indtil der findes et semikolon i karakterstrømmen. Herefter sammenlignes den streng, der er blevet modtaget, med de muligheder som boardet og protokollen stiller til rådighed (se afsnit 4.3.1). Sammenligningen er implementeret som en simpel if/else, der sammenligner strengene vha. strcmp-funktionen.

- Hvis strengen er 'XML', sendes først længden af XML-beskrivelsen efterfulgt af ':' (kolon), hvorefter hele XML-strengen sendes. Aflæsning af den serielle port startes på ny.
- Hvis strengen indeholder et kolon, splittes den op i to dele (navn og værdi).
    - Herefter sammenlignes strengen (navnet) med navnene på de sensorer/aktuatorer som enheden har, for at finde ud af om det er en gyldig forespørgsel.
    - Hvis der ikke er medsendt nogen værdi returneres den aktuelle værdi for sensoren. Aflæsning af den serielle port startes på ny.
    - Ellers sættes den medsendte værdi som den nye værdi for aktuatoren og aktuatorens værdi/tilstand returneres. Aflæsning af den serielle port startes på ny.
- Hvis strengen ikke kunne genkendes, returneres '`Error;`'.

4.4 Implementering af Arduino-framework

I dette afsnit beskrives implementeringen af et fælles softwareframework på Arduino-platformen, der implementerer kommunikationsprotokollen og gør det muligt at implementere enheder, der kan kommunikere med mobilapplikationen.

Det meste af koden er implementeret i en række metoder der er fælles for alle enheder. De steder hvor de enkelte enheder skal udvide koden, er beskrevet i de relevante afsnit.

4.4.1 Framework og protokoller

Denne del af implementeringen er fælles for alle enheder. Derudover vil der blive beskrevet den funktionalitet der er fælles, men hvor implementeringen kan afvige imellem enhederne.

Variable

Alle enheder har en række fælles variable der benyttes i programafviklingen. Der er implementeret to buffere, som benyttes til at læse fra den serielle port, og en anden buffer, der benyttes når XML-beskrivelsen skal sendes. Read-buffer størrelsen er defineret til 32 og XML-bufferen er 128 bytes.

```
char XMLCompare[] = "XML";
char errorString[] = "Error;";
char xml_buffer[XML_BUFFER_SIZE + 1];
char buffer[READ_BUFFER_SIZE];
char value[READ_BUFFER_SIZE];
```

Eksempel 23: Variable i frameworket

Herudover gemmes enhedens XML-beskrivelse i et const char-array som beskrevet i afsnit 6.1.2 *Overførsel af lange strenge*.

```
PROGMEM const char xml[] = "<XML>";
```

Hjælpemetoder

For at ensrette afviklingen af programmet på de enkelte enheder, er der blevet implementeret en række hjælpemetoder.

`reset_bt()`, der nulstiller Bluetooth-enheden ved at påvirke den reset-pin, der er koblet til enheden.

```
void reset_bt(){
  digitalWrite(RESET, HIGH);
  delay(10);
  digitalWrite(RESET, LOW);
  delay(2000);
}
```

Eksempel 24: Funktionen reset_bt()

`int startsWith(char* text, char* token)`, der undersøger om en streng starter med en anden streng. Funktionen returnerer '0' hvis '`text`' starter med '`token`', ellers returneres '-1'.

```
int startsWith(char* text, char* token) {
  int tokenLength = strlen(token);
  if(strlen(text) < tokenLength) {
    return -1;
  }
  for(int i = 0; i < tokenLength; i++) {
    if(text[i] != token[i]) {
      return -1;
    }
  }
  return 0;
}
```

Eksempel 25: Funktionen startsWith()

`clear_buffer()` og `clear_value()`, der tømmer henholdsvis buffer- og value-arrayene, der er nævnt tidligere, ved at sætte alle indgange til 0.

Setup

Denne metode findes i alle Arduino-programmer og indeholder den kode der skal køres når enheden starter op. Indholdet i metoden afhænger af den enkelte enhed, men en del af koden er fælles for alle eksterne enheder.

```
void setup() {
  pinMode(LED,OUTPUT);
  //Opsætning af andre digitale udgange
  xml_buffer[XML_BUFFER_SIZE] = '\0';
  Serial.begin(115200);
  reset_bt();
  clearBuffer();
  clearValue();
  //Anden enhedsspecifik opsætning
  digitalWrite(LED, HIGH);
}
```

Eksempel 26: Setup()-metoden på Arduino

Koden sørger for at de gængse elementer bliver initialiseret så enheden er klar til brug og indikerer at applikationen er startet ved at tænde for en LED.

Programløkke

Programløkken består af to hoveddele; en del der læser fra den serielle port og en del der håndterer modtagne kommandoer.

Den første del afvikles løbende i en løkke (loop()-metoden), der automatisk bliver eksekveret i en uendelig løkke af Arduino-enheden, mens den er tændt.

```
int val = Serial.read();
if (val != -1) {
  if (val == 59) { //59 er ;
    //Håndter kommando
  }
  else {
    buffer[recieved++] = val;
    if(recieved == READ_BUFFER_SIZE) {
          clearBuffer();
    }
  }
} else {
  delay(LOOP_WAIT);
}
```

Eksempel 27: Programløkke for læsning fra serielporten

Dette er en simpel implementering som læser den næste karakter på den serielle port og tilføjer den til read-bufferen. Hvis der findes et semikolon håndteres den fundne kommando. Hvis læsebufferen fyldes op med karakterer, uden at et semikolon er blevet fundet, tømmes bufferen. Der holdes en kort pause mellem læsninger på den serielle port hvis der ikke modtages en karakter.

Den anden del er en simpel sammenligning af modtagne kommandoer, med strenge der er gemt i programmet. Den eneste sammenligning der er fælles for alle enheder, er håndteringen af '`XML;`'-kommandoen, hvor XML-beskrivelsen sendes i små bidder af størrelsen ' `XML_BUFFER_SIZE`' bytes.

Andre kommandoer indsættes som en række af 'else if'-statements der sammenligner den modtagne kommando med en der er gemt i programmet, på samme måde som sammenligningen med strengen 'XML'.

```
if (strcmp(buffer, XMLCompare) == 0) {
  size_t xml_length = strlen_P( xml ) + 1;
  Serial.print( xml_length - 1 );
  Serial.print(":");
  int index = 0;
  while(xml_length > XML_BUFFER_SIZE) {
    strncpy_P(xml_buffer, &(xml[index]), XML_BUFFER_SIZE);
    Serial.print( xml_buffer );
    xml_length -= XML_BUFFER_SIZE;
    index += XML_BUFFER_SIZE;
  }
  strncpy_P(xml_buffer, &(xml[index]), XML_BUFFER_SIZE);
  Serial.print( xml_buffer );
}
//Sammenligning med andre kommandonavne
else {
  Serial.print( errorString );
}
delay( SHORT_WAIT );
clearBuffer();
clearValue();
```

Eksempel 28: Sammenligning af kommandostrenge

Ændring af aktuatorværdi og -tilstand

Ændringer af aktuatorværdier og -tilstande er lidt mere kompliceret end en simpel aflæsning af en værdi. Protokollen kræver at kommandoen, inklusiv den nye værdi, sendes på formatet 'sensor:værdi;' og den eksisterende sammenligningsløkke sammenligner med hele den modtagne streng. Derfor håndteres disse kommandoer ved hjælpe af følgende kodeeksempel:

```
else if(strchr(buffer, ':') != 0) {
  char* pos = strchr(buffer, ':');
  strcpy(value, ++pos);
  if(startsWith(buffer, SENSORNAVN1) == 0) {
    //Sæt sensornavn1 til værdien i value
   ...
    Serial.print('NY TILSTAND FOR SENSOR 1');
  }
  else if(startsWith(buffer, AKTUATORNAVN2) == 0) {
    //Sæt sensornavn2 til værdien i value
    ...
    Serial.print('NY TILSTAND FOR SENSOR 2');
  }
}
```

Eksempel 29: Ændring af aktuatorværdi

Først undersøges om strengen indeholder et kolon og hvis den gør, gemmes alt efter kolonet i værdi-bufferen. Læse-bufferen sammenlignes derefter med enhedens aktuatornavne for at finde den rigtige, ved at benytte 'startsWith'-metoden.

Herefter behandles den nye værdi/tilstand og aktuatorens aktuelle tilstand/værdi returneres.

4.5 Service discovery

I dette afsnit beskrives analysen og implementeringen af den service discovery-protokol, der gør det muligt for mobiltelefonen at finde de eksterne enheder.

4.5.1 Analyse af service discovery protokollen

Protokollen skal gøre det muligt for mobilapplikationen at søge efter eksterne enheder, som implementerer dette framework, og forbinde til dem uden at de kender hinanden på forhånd.

Dette kræver at de eksterne enheder stiller information til rådighed, som mobilapplikationen kan hente og behandle, der identificerer enheden som en del af en aktuel historie eller som en enhed der implementerer dette framework. Samtidig skal det være muligt for mobilapplikationen hurtigt at sortere imellem alle enheder der bliver fundet og sortere de irrelevante fra.

Bluetooth Service Discovery Protocol

En mulighed er at benytte den service discovery protokol der er indbygget i Bluetooth-protokollen. SDP giver Bluetooth-enheder mulighed for at forespørge hvilke services en anden Bluetooth-enhed stiller til rådighed. Services defineres ved et unikt service ID (der benyttes 128-bit UUID'er) for hver service, hvor der er prædefineret en række standard services. Det er muligt at definere egne services blot ved at tildele dem et unikt UUID.

Dette kan udnyttes til at lave et framework-specifikt service ID, som på Arduino-enheden synliggøres som en understøttet service. Mobilapplikationen kan så lede efter Bluetooth-enheder der har denne service, for at sortere de enheder fra der ikke understøtte frameworket.

Bluetooth service discovery er understøttet i Java ME's Bluetooth-klasser, igennem det DiscoveryAgent-objekt som også benyttes til at finde Bluetooth-enhederne med (se afsnit 4.3.2 *Implementering i mobilapplikationen*).

På Arduino-enheden tilføjes en ny service ved at sende en kommando til den fysiske Bluetooth-enheds iWrap protokolstak. Kommandoen sendes efter en reset af Bluetooth-enheden (under opstarten) og ser ud som følger.

```
reset_bt();
Serial.println("SDP ADD <UUID> <SERVICENAVN>");
```

Eksempel 30: Tilføjelse af ny bluetooth service på Arduino

Herefter er det muligt for mobilapplikationen at spørge en kendt Arduino-enhed, om denne har servicen '`SERVICENAVN`' med id'et '`UUID`'.

Friendly Name

Denne metode benytter sig af Bluetooth-enhedernes navn (kaldet friendly name[11]), der er synligt for alle Bluetooth-enheder (det er også muligt for en enhed ikke at have et navn, så er det kun adressen der er synlig). Ved at give enhederne et navn med et kendt præfiks, er det muligt for mobilapplikationen at genkende de enheder som understøtter frameworket. Aflæsning af friendly name er hurtigt og understøttet af alle Bluetooth-enheder.

Når applikationen finder en Bluetooth-enhed sammenlignes dennes navn med den kendte streng, og hvis den matcher tilføjes den til listen over fundne enheder. Ellers ignoreres enheden.

F.eks. kan et museum ('Nationalmuseet') præfikse deres enheder med strengen 'Nationalmuseet-' og lave deres applikation så den kun forbinde til enheder med dette præfiks.

Evaluering

Fordelene ved SDP er at det vil være muligt at finde understøttende enheder vha. eksisterende protokoller samt man altid vil kunne afgøre om en ekstern enhed understøtter det ønskede framework. Dette vil gøre det muligt at lave applikationer som kan benyttes til mange forskellige formål, uden at der skal tages hensyn til det specifikke formål. Applikationerne vil altid kunne finde enheder, der understøtter frameworket.

Ulemperne er at service discovery er langsomt og lægger et større overhead på søgningen efter eksterne enheder, specielt i omgivelser med meget Bluetooth-trafik. Hver gang en ekstern enhed findes skal der foretages et service discovery, hvilket tager tid. En anden ulempe er at alle applikationer i teorien understøtter alle enheder, hvilket ikke nødvendigvis er ønsket i en kommerciel implementering af dette system.

Friendly name-metoden er en simpel og hurtig måde at identificere de rigtige enheder på, der samtidig kan tilrettes så applikationer til et bestemt formål (f.eks. på et bestemt museum) kun finder enheder der passer til dette formål. Ulempen er at det netop ikke er generelt for hele frameworket og applikationer derfor nødvendigvis skal laves specielt til hvert formål.

En anden ulempe er simpliciteten i godkendelsesproceduren. Ved kun at kigge på et offentligt tilgængeligt navn, er det muligt at der kommer andre Bluetooth-enheder indenfor rækkevidde med samme navn. Dette vil gøre søgningen efter de rigtige enheder langsommere, da systemet vil antage at denne (falske) enhed også understøtter protokollen. Den tid det tager at opdage at enheden ikke er "ægte", er tid som brugeren oplever som ventetid før den rigtige enhed kommer frem. Dette kan delvist afhjælpes ved at huske enhedernes Bluetooth-id og holde en persistent liste over "falske" enheder, så de ikke forstyrrer søgningen mere end én gang.

De to metoder kan kombineres så 'friendly name' først benyttes til en hurtig screening af enheder, hvorefter SDP benyttes til at finde ud af om enheden reelt er en del af systemet. Dette vil give fordelene fra begge metoder, uden at ulemperne følger med.

[11] http://www.bluetooth.com/English/Technology/Pages/Glossary.aspx#N

4.5.2 Implementering

'Friendly name'-metoden implementeres som en simpel sammenligning med en kendt streng. I denne prototype er strengen hardcodet ind i programmet, men i en senere version kan strengen tilføjes som en property i XML-filen til dramaet eller som parameter til BluetoothModule-klassen, der står for søgningen efter Bluetooth-enheder (se afsnit 3.3.2 *Moduler til kommunikation og discovery*).

```
String name = arg0.getFriendlyName(true);
if (name.indexOf("SPECIALE-") != -1) {
  Device device = DeviceManager.createDevice(name.substring(7));
  device.setBTAddress(arg0.getBluetoothAddress());
  foundDevices.addElement(device);
}
```

Eksempel 31: Friendly name-sammenligning

Implementeringen af service discovery protokollen på Java ME-platformen har ikke fungeret. Det var ikke muligt at få mobilapplikationen til at genkende den specielle service som blev tilføjet Arduino-enhederne. Ved en søgning fra en Windows 7 laptop var det muligt at se servicen på den eksterne enhed.

Hvorfor Java ME-applikationen ikke kunne genkende servicen vides ikke, og metoden implementeres derfor ikke i prototypen. Implementeringen af SDP er et punkt som skal undersøges nærmere i en eventuelt konkret implementering af systemet.

4.5.3 Resultat

Det var ikke muligt at detektere den specialdesignede service på Arduino-enhederne, hvorfor service discovery i denne applikation kun benytter sig af 'friendly name'-metoden.

Herved bliver det muligt at identificere de enheder, der tilhører dette projekt (præfikset er 'SPECIALE'). Hvis en "falsk" enhed kommer indenfor rækkevidde, vil søgetiden for udstillingerne blive forlænget en smule, da systemet først forsøger at hente XML fra den "falske" enhed.

4.6 Opsummering

Konfigurationsprotokollen (fra Kapitel 3) er blevet implementeret på Arduino-platformen som beskrevet i afsnit 4.2 (*Konfigurationsprotokol på Arduino*). Kommunikationsprotokollen er blevet implementeret både på Java ME (4.3.2) og Arduino-platformene (4.3.3).

Ligeledes er der blevet implementeret en service discovery protokol (4.5) og et softwareframework til Arduino-platformen (4.4).

Implementeringen besvarer følgende af specialets delspørgsmål:

Delspørgsmål III - *Hvordan håndteres fysisk interaktion med indlejrede enheder fra en mobiltelefon?*

Delspørgsmål IV - *Hvordan skabes den fysiske interaktion igennem et interface til indlejrede enheder?*

Spørgsmålene er blevet besvaret igennem implementeringen af følgende elementer.

Der er blevet implementeret en simpel og effektiv kommunikationsprotokol, som gør det muligt for mobilapplikationen og de eksterne enheder at kommunikere med hinanden. Protokollen gør det muligt for mobilapplikationen at forespørge efter en XML-beskrivelse af enheden, aflæse sensorværdier og ændre værdi/tilstand af aktuatorer, som beskrevet i afsnit 1.4.1 (*Krav til frameworket*).

Implementeringen af konfigurationsprotokollen på Arduino, samt koblingen mellem denne og kommunikationsprotokollen, opfylder følgende af specialets forventede resultater:

A3 (*Et interface til at aflæse og styre sensorer/aktuatorer, der er koblet til de indlejrede* enheder)

A4 (*Et interface til dynamisk at udvide en eksisterende historie, på mobiltelefonen, ved hjælp af en beskrivelse hentet fra en indlejret enhed*)

Service discovery protokollen opfylder krav til denne (afsnit 1.4.1) på trods af at der ikke benyttes Bluetooth service discovery (SDP) til dette. Kravet til protokollen er ”*Opdagelsen af, og oprettelsen af forbindelse til, en ekstern enhed, skal foregå transparent for brugeren. Frameworket skal gøre det muligt at lede efter, finde og forbinde til enheder løbende, mens en applikation er aktiv.*”, hvilket implementeringen til fulde opfylder.

Kapitel 5. Prototype og verifikation

I dette kapitel beskrives den proof-of-concept prototype, der implementeres af den i specialet fremsatte løsning. Prototypen har til formål at undersøge og verificere den implementering, der er beskrevet i afsnit 3.3 (*Konfigurationsprotokol*) og 4.3 (*Kommunikationsprotokol*). Implementeringen undersøges i forhold til specialets formål, forventede resultater og de analyser der er foretaget i afsnit 3.2 og 4.3.1.

Prototypen tager udgangspunkt i en historie baseres på en rundvisning på et museum, hvor der benyttes Arduino-enheder til at formidle information om kunstværkerne og til at skabe interaktion med udstillingerne.

5.1 Historie

Først designes og beskrives der en historie, der opfylder de krav der er stillet, samt en beskrivelse af hvilke sensorer og aktuatorer der benyttes i historien. Herefter implementeres historien som en XML-beskrivelse der følger den konfigurationsprotokol, der er beskrevet i afsnit 3.2.

5.1.1 Design af historien

Prototypehistorien implementeres som en kort rundvisning på et fiktivt museum. Museet har en række forskellige udstillinger, der alle benytter det nye system til at skabe interaktion mellem brugeren og udstillingerne. Hver udstilling består af en fysisk komponent, der matcher det der ville være i et rigtigt museum, og en skjult komponent, den indlejrede enhed, der styrer interaktionen med den fysiske udstilling. Brugeren kan frit vælge rækkefølgen af udstillingerne, om nogle skal ses flere gange eller om en udstilling skal springes helt over.

Der er i alt tre forskellige interaktive udstillinger; En udstilling om Van Gogh, en udstilling med et motorstyret miniakvarium (baseret på et MiniQuarium[12]) og en udstilling af sensorer der giver brugeren informationer om det rum, enheden befinder sig i.

Enkelte udstillinger kan være afhængige af ressourcer, der skal downloades før selve udstillingen kan starte. Ressourcerne er angivet i XML-beskrivelsen og download af dem foretages automatisk i baggrunden mens brugeren venter.

XML-beskrivelserne til museumshistorien og de interaktive knuder kan ses i appendikset afsnit 9.2.1. Designet af de enkelte beskrivelser er beskrevet herunder.

Museumsapplikation

Selve applikationen, som skal installeres på mobiltelefonen, er et simpelt MOURDA-baseret drama, der indeholder enkelte historieknuder.

[12] Et kunstigt batteridrevent miniakvarium med to plastikgopler i, der svømmer rundt når motoren tændes. Lyset i akvariet kan også tændes og slukkes.
http://www.brainstormltd.co.uk/aspx/products.aspx?cat=FACTFINDERS&code=EXPLORE%20And%20LEARN

Udover velkomstskærmen, der byder brugeren velkommen til museet, har historien kun en enkelt historieknude, nemlig en hovedmenu. Fra denne menu har brugeren mulighed for at starte rundvisningen eller afslutte programmet.

Hovedmenuen indeholder også en eventhandler, der håndterer visningen af en guide til hvordan brugeren benytter applikationen. Denne guide vises så længe brugeren bevæger sig rundt på museet og der ikke er fundet eksterne enheder.

Derudover indeholder historien en række globale historieelementer. Udover Bluetooth-modulet er der globale eventhandlers, der håndter en afsluttet Bluetooth-søgning. Hvis der ingen enheder er fundet vendes der tilbage til hovedmenuen (der igen viser guiden til brugeren), ellers vises der en valgskærm til brugeren (se delafsnittet *Modul og event* i afsnit 3.3.2 for en beskrivelse af Bluetooth-modul og events).

Hvis brugeren svarer nej til at se denne udstilling, eller hvis brugeren allerede har set den, vises i stedet en skærm over de udstillinger der er indenfor rækkevidde af telefonen. Brugeren kan så enten fortsætte søgningen (ved ikke at foretage sig noget) eller vælge at se en udstilling.

XML-beskrivelsen af hoveddramaet kan ses i appendikset afsnit 9.2.1 delafsnit Museumsdrama.

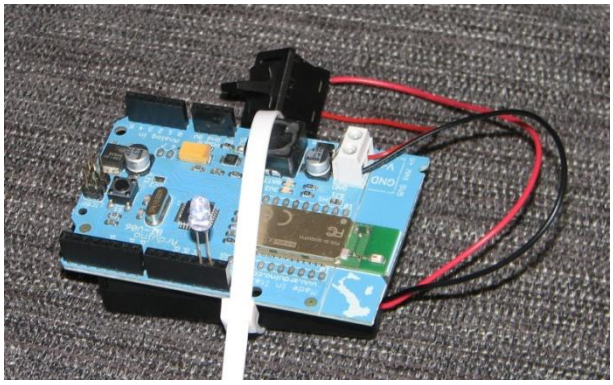
Van Gogh

Ideen med denne udstilling er at brugeren, hvis denne ønsker det, kan få flere oplysninger om en kunstner direkte på telefonen. F.eks. i en situation hvor brugeren står foran et maleri af en bestemt kunstner.

Interaktionen er implementeret som en standard historieudvidelse, der indeholder tre skærme som hver viser et maleri af Van Gogh og en kort beskrivelse af maleriet.

Før selve udstillingen kan starte er det nødvendigt at downloade ressourcer dertil (de tre billeder der skal vises, de fylder i alt ca. 40 kB).

Formålet med denne udstilling er at vise et eksempel på en dynamisk historieudvidelse, i et praktisk brugsscenarie.



Figur 49: Van Gogh-udstillingen

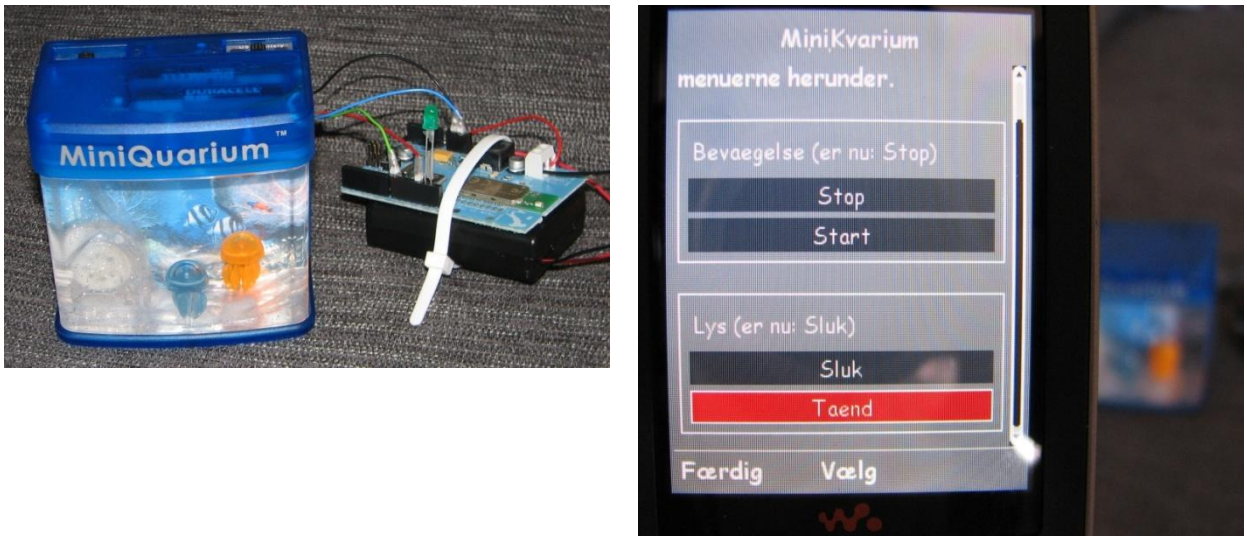
XML-beskrivelsen til Van Gogh-enheden kan ses i appendikset afsnit 9.2.1 delafsnit Van Gogh.

MiniKvarium

MiniKvarie-udstillingen består af MiniKvariet, der er forbundet til den indlejrede enhed, hvor det kun er den forreste skærm af MiniKvariet der skal være synlig for brugeren. Brugeren har ikke nogen direkte fysisk mulighed for interaktion med udstillingen, men får i stedet vist mulighederne på telefonen.

Udstillingen er implementeret som to knuder, hvor den ene er en introduktionsskærm og den anden er en dynamisk komponent, der indeholder to aktuatorer. Disse aktuatorer svarer til de muligheder som MiniKvariet rent fysisk stiller til rådighed (lys- og motorstyring).

Formålet med udstillingen er at vise et eksempel på visualisering og styring af tilstandsaktuatorer på en ekstern enhed (se delafsnit *Aktuator* i afsnit 3.3.4).



Figur 50: MiniKvarium-udstillingen

XML-beskrivelsen til MiniKvariet kan ses i appendikset afsnit 9.2.1 delafsnit MiniKvarium.

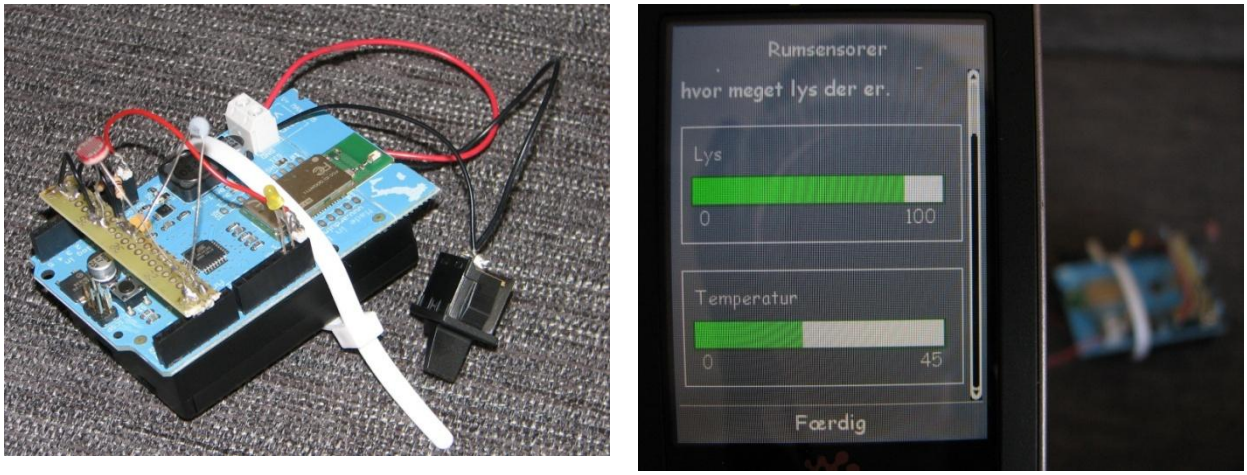
Rumsensorer

Den sidste udstilling består af en enhed hvor der stikker to sensorer ud fra; en lyssensor og en temperatursensor. Det er op til brugeren selv at interagere med enheden ved at påvirke de to sensorer (enten ved berøring af sensorerne eller ved at flytte rundt på enheden).

Denne interaktion er implementeret som en enkel dynamisk knude, der indeholder to sensorer. Sensorerne bliver automatisk opdateret af telefonen hver femte sekund, for at give brugeren en løbende opdatering af ændringer der foretages i miljøet. Det er også muligt for en bruger at forholde sig passivt og blot lade andre gæste/brugere påvirke sensorerne, værdierne vil stadig blive opdateret i mobilapplikationen.

Formålet er at vise et eksempel på hvordan det er muligt at lade sensorpåvirkninger i det fysiske rum påvirke interfacet på telefonen i real-time.

Udstillingen kan ses nedenfor i Figur 51.



Figur 51: Rumsensor-udstillingen

XML-beskrivelsen til rumsensorenheden kan ses i appendikset afsnit 9.2.1 delafsnit Rumsensor.

5.2 Mobilapplikation

Historien implementeres på mobiltelefonen, for at verificere at implementeringen af protokolhåndteringen lever op til de krav der er stillet.

Kildekoden til mobilapplikationen og kompilerede udgaver af programmet, kan findes på den vedlagte CD eller på nettet på http://speciale.barmonger.org

5.2.1 Implementering

Mobilapplikationen er blevet implementeret i de forrige afsnit (konfigurationsprotokollen i afsnit 3.3, kommunikationsprotokollen i afsnit 4.3.2) og det er ikke nødvendigt at implementere ny funktionalitet i applikationen, for at understøtte de funktioner som prototypehistorien benytter sig af.

Dramaet, der er beskrevet i forrige afsnit, kompileres ind i applikationen og afvikles automatisk når programmet starter.

Teststub

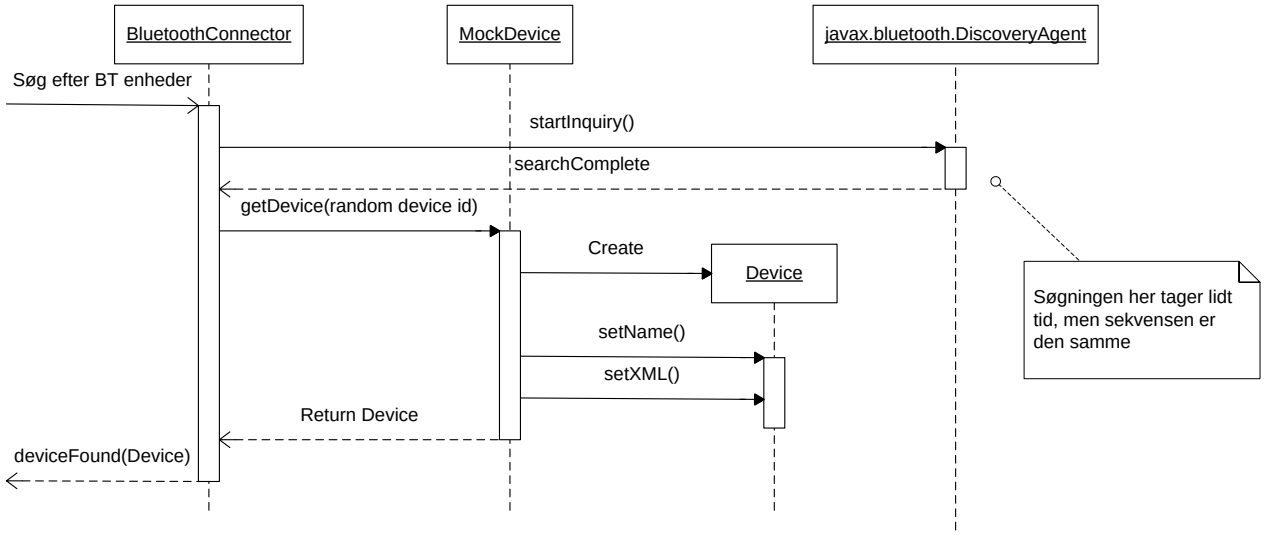
For at implementeringen kan testes korrekt på mobiltelefonen, pakkes Bluetooth-kommunikationen ind i en teststub, der benyttes i stedet for de fysiske enheder. Derved opnås at det bliver muligt at teste mobilapplikationens implementering af konfigurations- og kommunikationsprotokollen, uden at være afhængig af en korrekt implementering på Arduino-enhederne.

Stubben implementeres i klassen MockDevice, der udvider klassen Device (se afsnit 3.3.2 *Moduler til kommunikation og discovery*). Stubben overloader alle metoderne i Device-klassen med metoder, der simulerer en langsom Bluetooth-forbindelse men ellers opfører sig som et rigtigt Device.

MockDevice-klassen indeholder XML-beskrivelser for alle tre eksterne enheder (fra afsnit 5.1) og simulerer opførslen af dem alle. Den har en statisk getDevice()-metode, der benyttes i stedet for en constructor, til at hente det Device ud der er brug for.

BluetoothConnector-klassen, der står for at finde fysiske Bluetooth-enheder og pakke dem ind i Device-objekter, benytter MockDevice-klassen når en Bluetooth-søgning er afsluttet. I stedet for af lede efter eksterne enheder, returneres der tilfældige MockDevices til resten af systemet (Se Figur 47 i delafsnittet *BluetoothConnector* i afsnit 4.3.2, for en beskrivelse af BluetoothConnector-klassens opførsel).

Figur 52 viser sekvensen i en Bluetooth-søgning hvor MockDevice benyttes til at lave et Device.



Figur 52: Brug af MockDevice i Bluetooth-søgning

Samtidig logger MockDevice-klassen alle metodekald (og parametre), der foretages igennem Device-klassens interface, for bagefter at kunne verificere hvorvidt mobilapplikationen opfører sig som forventet i forhold til eksterne enheder.

Teststubben benyttes ikke i selve verifikationen af prototypen men kun til at verificere at implementeringen på mobilplatformen følger de opstillede krav. I verifikationen af det samlede system benyttes de klasser, der er nævnt i implementeringen i afsnit 3.3.2.

5.2.2 Verifikation og test

Testen af mobilapplikationen foretages ved at programmet installeres på en mobiltelefon og afvikles der. Opførslen af programmet dokumenteres, både tekstuelt og visuelt, og sammenlignes med den forventede opførsel.

Skærmbilleder af programafviklingen er et af de vigtigste redskaber til sammenligning af programmets opførsel med forventede resultater og verifikationen vil hovedsageligt basere sig på disse. Loggen over metodekald sammenlignes også med den forventede opførsel af programmet. Testen godkendes hvis opførslen svarer til krav og forventede resultater.

I afsnittet herunder bliver resultaterne af testen og verifikationen diskuteret, opdelt i fire dele; hovedprogrammet og de tre indlejrede enheder.

I appendiks afsnit 9.4 *Verifikation og test af mobilapplikationen* er det muligt at se skærmbilleder fra testen samt en beskrivelse af hvordan testen blev udført.

Programmet

Første trin i verifikationen er at undersøge hovedprogrammets opførsel ved kun at se på den generelle opførsel, der er uafhængig af de specifikke eksterne enheder.

Testen viste at programmet opfører sig som forventet og de skærme der er beskrevet i XML-beskrivelsen bliver vist korrekt.

Van Gogh

Van Gogh-udstillingen er implementeret som en række simple billedskærme, der indeholder en beskrivende tekst. Brugeren har mulighed for at navigere videre til næste billede og ved sidste billede er der mulighed for at gå tilbage til menuen.

Testen viste at XML-beskrivelsen blev hentet fra den eksterne enhed og indlæst korrekt og brugeren får mulighed for at se de forskellige billeder med beskrivende tekst.

Loggen fra MockDevice-klassen viser at systemet har kaldt følgende metoder derpå:

1. *getXML for device VanGogh*

Dette stemmer godt overens med at der kun skal være foretaget et enkelt kald, til den eksterne enhed, for at hente XML-filen.

MiniKvarium

MiniKvarium-enhedens beskrivelse indeholder to elementer. En introskærm og en dynamisk knude. Introskærmen til MiniKvariet er en simpel billedskærm med tekst, der vises til brugeren når udstillingen startes. Der er mulighed for at brugeren kan navigere til den interaktive del af udstillingen. Den dynamiske knude indeholder to aktuatorer som giver brugeren mulighed for at interagere med MiniKvariet.

Testen viste at beskrivelsen blev hentet korrekt fra den eksterne enhed, samt at brugergrænsefladen blev genereret korrekt på mobiltelefonen. Det var muligt at interagere med begge aktuatorerne og både tænde og slukke for lyset og motoren (bevægelsen i akvariet). Samtidig blev den aktuelle tilstand af aktuatorerne automatisk opdateret i brugergrænsefladen.

Loggen fra MockDevice ser ud som følger:

1. *getXML for device MiniQuarium*
2. *getValue for sensor pump for device MiniQuarium*
3. *getValue for sensor light for device MiniQuarium*
4. *setState for actuator light with state 1 for device MiniQuarium*
5. *setState for actuator light with state 2 for device MiniQuarium*

6. *setState for actuator pump with state 1 for device MiniQuarium*
7. *setState for actuator pump with state 2 for device MiniQuarium*

Ud fra loggen ses det at de aktuelle værdier hentes fra enheden som det første [2 - 3]. Herefter sættes en ny tilstand på lyset (light) hvor der først skiftes til '1' [4] og derefter tilbage til '2' [5]. Dette gentages for 'bevægelse' (pump) [6-7].

Denne sekvens af metodekald svarer til den forventede opførsel ved aktuatorenheder.

Rumsensor

Rumsensorudstillingen har ingen introskærm og den eneste knude, der er angivet i XML-beskrivelsen, er den interaktive knude. Denne knude indeholder to sensorer, som hver svarer til en fysisk sensorenhed på Arduino-boardet.

Testen viste at beskrivelsen blev hentet og indlæst korrekt og at kommunikationen med sensorerne fungerede som forventet. Sensorværdierne blev automatisk opdateret i brugergrænsefladen.

Loggen fra MockDevice viser følgende metodekald:

1. getXML for device Sensor
2. getValue for sensor light for device Sensor
3. getValue for sensor temp for device Sensor
4. getValue for sensor light for device Sensor
5. getValue for sensor temp for device Sensor
6. getValue for sensor light for device Sensor
7. getValue for sensor temp for device Sensor
8. getValue for sensor light for device Sensor
9. getValue for sensor temp for device Sensor
10. …

Sekvensen af metodekald viser at applikationen korrekt kalder getValue() for hver af sensorerne kontinuerligt.

5.3 Arduino-prototype

Historien implementeres både i software og hardware på Arduino-enhederne og implementeringen testes eksperimentelt for at verificere at opførslen af enhederne svarer til kravene i protokolbeskrivelsen.

Den præcise implementering af Arduino-enhederne er meget enhedsspecifik og detaljerne ligger udenfor specialets formål. En beskrivelse af implementeringen kan ses i appendikset afsnit 9.3 *Arduino-implementering*.

Kildekoden til Arduino-enhederne kan ses på den vedlagte CD (i mappen `/Arduino`) eller på nettet på http://speciale.barmonger.org/arduino.php

5.3.1 Verifikation og test

De forskellige Arduino-enheder er blevet samlet og hardwaren er blevet testet manuelt. I dette afsnit testet den implementerede software på Arduino-enhederne i samspil med den fysiske hardware, i forhold til de krav der er opstillet til enhedens opførsel. Testen verificerer at kommunikationsprotokollen er implementeret korrekt (afsnit 4.3.3 *Implementering på Arduino-platformen*), at indholdet i konfigurationsprotokollen (afsnit 4.2 *Konfigurationsprotokol på Arduino*) matcher de fysiske navne på sensorer/aktuatorer samt at det tilkoblede hardware opfører sig som enheden beskriver.

Testen udføres vha. Arduino-værktøjests serielmonitor, der benyttes til at oprette en Bluetooth-forbindelse til enhederne. Først hentes XML-beskrivelsen fra enheden og den sammenlignes med den forventede XML-beskrivelse.

Herefter verificeres de sensorer, der er beskrevet i XML'en, ved at aflæse deres værdi. Hvis det er muligt foretages der flere målinger af sensoren mens denne påvirkes, for at afprøve hvorvidt systemet kan håndtere input.

Aktuatorer verificeres på samme måde som sensorerne. Tilstandsaktuatorer testes ved at afprøve alle tilstande, værdiaktuatorer testes ved at prøve forskellige værdier i området. Opførslen af den fysiske hardware, som styres af aktuatoren, benyttes samtidigt til at verificere om aktuatoren er implementeret korrekt.

VanGogh

Denne enhed har kun en XML-beskrivelse der skal verificeres.

Kommando	Påvirkning	Observeret opførsel	Forventet opførsel	Returværdi	Forventet værdi
XML;	-	-	-	3317: <XML-beskrivelsen>	3317: <XML-beskrivelsen>

Tabel 4: Test af VanGogh-enheden

Den returnerede XML-beskrivelse matchede den forventede og længden 3317 svarer til antallet af bytes i beskrivelsen.

MiniKvarium

Denne enhed har, udover XML-beskrivelsen, to tilstandsaktuatorer ('pump' og 'light') der kan påvirkes.

Kommando	Påvirkning	Observeret opførsel	Forventet opførsel	Returværdi	Forventet værdi
XML;	-	-	-	2750: <XML-beskrivelsen>	2750: <XML-beskrivelsen>
light;	-	Lys slukket	Lys slukket	2	2
pump;	-	Motor slukket	Motor slukket	2	2
light:1;	-	Lys tændt	Lys tændt	1	1
light:2;	-	Lys slukket	Lys slukket	2	2
pump:1;	-	Motor tændt	Motor tændt	1	1
pump:2;	-	Motor slukket	Motor slukket	2	2

Tabel 5: Test af MiniKvarium

XML-beskrivelsen matcher den forventede og størrelsen svarer til antallet af bytes i beskrivelsen.

De to aktuatorer, lys og motor, opfører sig som forventet i forhold til de afsendte kommandoer. Lyset og motoren på det fysiske akvarium reagerede i forhold til de kommandoer der blev sendt.

Rumsensor

Denne enhed har, ud over mulighed for at hente XML-beskrivelsen, to sensorer som kan påvirkes: lyssensor (light) og temperatursensor (temp). Enheden placeres som udgangspunkt på et almindeligt kontor.

Kommando	Påvirkning	Observeret opførsel	Forventet opførsel	Returværdi	Forventet værdi
XML;	-	-	-	1821: <XML-beskrivelsen>	1821: <XML-beskrivelsen>
temp;	-	-	-	19	~20
light;	-	-	-	60	~50
temp;	Sensoren håndvarmes	-	-	31	~35
temp;	Sensoren køles ned i et køleskab	-	-	8	~5-10
light;	Sensoren stilles under en lampe	-	-	92	90-100
light;	Sensoren stilles i et mørkt rum	-	-	0	0-20

Tabel 6: Test af rumsensor

XML-beskrivelsen har den forventede størrelse og matcher den forventede beskrivelse.

Lyssensoren opfører sig som forventet og resultaterne er indenfor acceptable afvigelser. Enheden reagerede som forventet på alle '`light;`'-kommandoer;

Temperatursensoren opfører sig som forventet. Udgangstemperaturen var omkring stuetemperatur, køleskabet ca. 8 grader og håndtemperaturen lige over 30. Enheden reagerede som forventet på alle '`temp;`'-kommandoer.

5.4 Det samlede system

Efter at have verificeret at de to delelementer fungerer som forventet og opfylder de nødvendige krav, sættes de sammen til et samlet system. Herefter undersøges det om det samlede system stadig opfører sig som forventet.

5.4.1 Fremgangsmåde

Mobilapplikationen installeres på en mobiltelefon (SonyEricsson W910), som ikke tidligere har haft applikationen installeret. De tre eksterne enheder samles og klargøres. Herefter startes applikationen på mobiltelefonen og rundvisningen startes. Det verificeres at brugergrænsefladen opfører sig som i verifikationen af denne (se delafsnit *Programmet* i afsnit 5.2.2).

Herefter testes de tre eksterne enheder, sammen med mobiltelefonen, en af gangen. For hver enhed verificeres det at opførslen af mobiltelefonen matcher den forventede samt at brugergrænsefladen ser ud som forventet. Det verificeres også at de eksterne enheder opfører sig som forventet, i de tilfælde hvor der er sensorinput eller aktuatorer forbundet.

Testen udføres ved at en enhed tændes og der ventes til mobilen har fundet og forbundet til den og brugergrænsefladen er generet. Herefter udføres de samme testskridt som i den oprindelige verifikation af enheden (i afsnit 5.2.2 og 5.3.1). Systemets opførsel sammenlignes med de forventede, og tidligere observerede, resultater. Herefter afsluttes enheden (fra mobiltelefonen), den fysiske enhed slukkes og en ny tændes. Der testes igen.

5.4.2 Resultat

Testen viste at det samlede system opførte sig som forventet. Alle kravene stillet til delelementerne blev stadig opfyldt og resultaterne fra verifikationerne i afsnit 5.2.2 og 5.3.1 blev også opnået med det samlede system.

I forhold til de eksterne enheder, og interaktionen med dem, fungerede alt som forventet. Det var muligt at hente nye historieknuder og afvikle dem, og derved udvide (eller ændre) den igangværende historie (Van Gogh-enheden). Det var også muligt at hente en beskrivelse af sensorer, genere en brugergrænseflade og hente værdier fra sensorerne (Rumsensorer-enheden). Det samme galt for aktuatorerne, hvor det var muligt at forbinde til enheden og visualisere aktuatorerne, samt at ændre deres tilstand igennem interfacet (MiniKvarium-enheden).

Søgningen efter enhederne fungerede automatisk og enhederne fandt hinanden uden problemer.

For billeder og video det samlede system, se appendikset afsnit 9.1.2 (*Billeder*) og 9.1.3 (*Video*).

5.5 Opsummering

Prototypeimplementeringen havde til formål at være en proof-of-concept implementering af den løsning, der er blevet beskrevet i de forrige afsnit, samt at verificere at implementeringen af protokollerne (konfigurations-, kommunikations- og service discovery protokollerne) er brugbar i en praktisk implementering af hele systemet.

Proof-of-concept implementeringen skulle samtidig opfylde to af specialets forventede resultater:

B1 - *Det skal være muligt at starte en historie og udvide denne, ved at interagere med indlejrede enheder*

B2 - *Det skal være muligt at interagere med fysiske sensorer og aktuatorer, via en igangværende historie*

Testen af prototypen på mobilplatformen (se afsnit 5.2.2) viste at implementeringen af denne opfylder de krav der blev stillet til den. Ligeledes viste testen af Arduino-prototypen at denne opfylder de opstillede krav (se afsnit 5.3.1).

Den samlede løsning viste at de enkelte delelementer fortsat opfører sig som forventet, når de sættes sammen til et samlet system. Alle de eksterne enheder fungerede korrekt og mobilapplikationen håndterede visning af de grafiske brugergrænseflader og kommunikation med enhederne, som forventet.

I forhold til de forventede resultater viser prototypen at det er muligt at udvide en aktuel historie ved at hente nye historieknuder fra en indlejret enhed, samt at det er muligt at vende tilbage til den oprindelige historie efter afvikling af de nye knuder (*B1*).

Prototypen viste ligeledes at interaktion med fysiske enheder (sensorer og aktuatorer) er mulig i en aktuel historie, selvom denne ikke kender til enhederne, ved at benytte det i specialet fremlagte framework (*B2*).

De forventede resultater er derfor opnået med prototypen.

Kapitel 6. Evaluering og refleksion

I dette afsnit vil resultaterne af de forrige afsnit blive diskuteret og evalueret. Derudover indeholder afsnittet beskrivelser af mulige løsninger på de problemstillinger, der er blevet identificeret under implementeringen.

6.1 Evaluering af implementering

Dette delafsnit evaluerer implementeringsresultaterne fra Kapitel 3 (*Mobilapplikation og konfigurationsprotokol*) og Kapitel 4 (*Indlejrede enheder og kommunikationsprotokol*), opdelt i fire punkter der omhandler forskellige dele af specialet; protokollerne, Arduino-platformen, mobilapplikationen og det samlede system

6.1.1 Protokoller

Protokollerne der benyttes i frameworket er implementeret i forhold til de analyser der er foretaget i de respektive afsnit (3.2, 4.3 og 4.5).

Konfigurationsprotokol

Konfigurationsprotokollen er blevet implementeret så den løser de problemstillinger der blev identificeret i analysen (afsnit 3.1 og 3.2.1) samt de krav der blev stillet til systemet i Kapitel 1.

Se afsnit 3.4 (*Opsummering*) for en opsummering af resultaterne af implementeringen af konfigurationsprotokollen.

Protokollen gør det muligt at konfigurere brugergrænsefladen på den mobile enhed i forhold til de egenskaber som den eksterne enhed har. Det er muligt at håndtere søgning efter de eksterne enheder, indlæsning af globale ressourcer og skift mellem nye og eksisterende knuder i dramaet.

Dette giver tilsammen en komplet løsning der, som prototypeimplementeringen viser, gør det muligt at håndtere visning af og interaktion med, både sensorer og aktuatorer på de eksterne enheder, via en simpel autogenereret brugergrænseflade. Brugergrænsefladen kan håndtere et vilkårligt antal sensorer og aktuatorer, enten ved at de vises som en lang liste, eller der implementeres forskellige knuder til de forskellige elementer. Det er muligt at lave en brugergrænseflade hvor brugeren kan skifte mellem sensorer og aktuatorer, blot ved at angive en række events og kommandoer som brugeren kan vælge.

Kommunikationsprotokol

Kommunikationsprotokollen er blevet implementeret så den opfylder de krav der blev stillet til interaktionsmulighederne i systemet (se afsnit 1.2 og 1.6). Samtidig giver implementeringen en løsning på de problemstillinger, der blev identificeret i analysen (se afsnit 4.3.1).

Se afsnit 4.6 (*Opsummering*) for en opsummering af resultaterne af implementeringen af kommunikationsprotokollen.

Protokollen gør det muligt for mobilapplikationen at hente XML, aflæse sensorværdier og ændre aktuatortilstande. Implementeringen er enkel og protokollen er baseret på simple kommandoer, der sendes til den eksterne enhed. Alle kommandoer og returværdier sendes som en strøm af ASCII-karakterer over en emuleret serielport via Bluetooth.

På trods af den simple protokol er det muligt at interagere med alle typer af fysiske enheder, så længe disse kan kobles til, og styres af, en Arduino-enhed. Dette gælder for både analoge sensorer samt alle typer af enheder der kan styres/aflæses ved hjælp af Arduino-boardets digitale ind-/udgange.

Der kan kun være én mobilapplikation tilsluttet en Arduino-enhed af gangen, da der i protokollen ikke tages hensyn til om afsenderen af en kommando og modtageren af returværdien er den samme enhed. Data modtages og skrives blot på den serielle port. Se delafsnittet '*Flere/mange samtidige brugere*' herunder for en diskussion af dette problem.

Service discovery

Service discovery-protokollen blev implementeret sammen med kommunikationsprotokollen. Den opfylder de krav, der blev stillet i afsnit 1.4.1, dog uden at have implementeret en automatisk genkendelse af eksterne enheder ud fra Bluetooth-service discovery protokollen (se afsnit 4.5.3 *Resultat*).

Protokollen gør det muligt for en mobilapplikation at finde de eksterne enheder, der hører til det implementerede system. Dette gøres ved at det synlige navn på enheden er præfikset med en bestemt streng (i dette speciale er præfikset 'SPECIALE') og mobilapplikationen leder så efter enheder der har dette præfiks i navnet.

Protokollen er simpel og giver mulighed for hurtigt at finde de korrekte eksterne enheder. Men simpliciteten gør samtidig at applikationen ikke med sikkerhed kan vide om enheden er en "rigtig" enhed der understøtter frameworket, eller blot en Bluetooth-enhed med et matchende navn (se afsnit 4.5.2 *Implementering* for en beskrivelse af implementeringen af denne sammenligning).

6.1.2 Udvidelse af konfigurationsprotokol

Implementeringen af prototypen er baseret på de fysiske enheder der benyttes i specialet og de muligheder som de stiller til rådighed for mobilapplikationen. Derfor blev der implementeret eksterne enheder som gjorde det muligt at interagere med fysiske sensorer og aktuatorer. Mobilapplikationen stod så for at generere et fornuftigt brugerinterface til at interagere med disse sensorer og aktuatorer og hele softwareframeworket understøtte disse muligheder.

Samtidig er prototypen en simpel løsning, der ikke umiddelbart giver mulighed for kompliceret historieinteraktion eller baggrundsinteraktion med eksterne enheder. Al interaktion er synligt og håndteres ved hjælp af knapper i interfacet.

Sensor- og aktuatorbegreberne kan umiddelbart benyttes til at repræsentere en række forskellige interaktionselementer (f.eks. kan en tilstandsaktuator benyttes som en komponent hvor brugeren skal træffe et valg), men semantikken i XML-beskrivelsen skal helst matche den forventede funktion. Hvis

dette framework skal benyttes til mere generelle systemer med mange forskellige former for input og output, vil det være nødvendigt at udvide konfigurationsprotokollen så den giver disse muligheder.

En eventuel løsning af denne problemstilling er blevet fremstillet i afsnit 3.2.4 *Diskussion af udvidelser* af konfigurationsprotokol. Løsningen gør brug af en række abstrakte højniveau primitiver i stedet for de implementeringsspecifikke Sensor- og Aktuatorobjekter.

Evaluering

Implementeringen af en mere abstrakt og generel funktionalitetsbeskrivelse (som beskrevet i afsnit 3.2.4) vil betyde at konfigurationsprotokollen er nemmere at udvide på et senere tidspunkt, samt at udvidelser vil være mere bagud-kompatible med tidligere versioner af protokollen.

Ved ikke at binde sig op på implementeringsdetaljer bliver det muligt at udtrykke meget mere kraftfulde interaktioner med den samme protokol. F.eks. i situationer hvor applikationen kan afvikles på mere end én platform eller hvor der er flere versioner af samme applikation, der skal kunne håndtere den samme eksterne enhed.

Derudover er det samtidig et svar på et af specialets delspørgsmål:
Delspørgsmål V *Hvordan udvides prototypeimplementeringen med andre former for interaktion og dynamisk historiefortælling?*

Prototypeimplementeringen kan udvides med andre former for interaktion og dynamik, ved at benytte sig af mere generelle og abstrakte funktionalitetskomponenter i konfigurationsprotokollen.

6.1.3 Arduino-platformen

Implementeringen af Arduino-platformen er foretaget så den opfylder de krav til kommunikationsprotokollen der er opstillet i afsnit 4.3.3. Implementeringen har resulteret i en simpel framework-opbygning, der gør det muligt hurtigt at implementere en ny ekstern enhed til et system, og håndtering af konfigurationsprotokollen er indbygget i det. Frameworket er implementeret så det kan afvikles på alle Arduino-enheder, der understøtter Bluetooth.

I dette delafsnit vil der blive diskuteret en række problemstillinger, der er blevet identificeret i forbindelse med implementering og verificering af Arduino-platformen.

Rækkevidde af Bluetooth

Bluetooth-enhederne, på de Arduino-boards der benyttes i specialet, er klasse 1-enheder med en teoretisk rækkevidde på op til 100 meter[13]. Dette giver problemer i situationer hvor der er flere enheder indenfor den samme afstand eller hvor knuden skal aktiveres når en bruger er tæt på den, f.eks. ved en udstilling på et museum. En rækkevidde på 100 meter vil formentlig give problemer i alle scenarier hvor dette framework benyttes.

Derfor skal der i en praktisk implementering af et system, baseret på dette framework, vælges indlejrede enheder der har en kortere rækkevidde, f.eks. klasse 2-enheder (ca. 10 meter) eller klasse 3-enheder (ca. 1 meter), for at undgå overlap mellem enheder der ikke er fysisk tæt på hinanden.

[13] http://bluetooth.com/English/Technology/Pages/Basics.aspx

Overførsel af lange strenge

XML-beskrivelserne på Arduino-enhederne kan blive meget lange og nemt fylde mange Kb. Som nævnt i afsnit 4.1.1 (*Hardware*) har de enheder, der benyttes i prototypen, kun 16Kb flash og 1Kb ram til rådighed. Derfor kan der potentielt opstå problemer under håndteringen af meget lange XML-beskrivelser. For at afhjælpe dette problem er der blevet implementeret to ændringer i protokollen og på Arduino-platformen.

Arduino-platformen indlæser som standard alle erklærede variable i chippens ram, da det er den hurtigste lagringsmulighed. Men da længden på XML'en kan overskride 1Kb (og da resten af programmets variable også er gemt i ram'en betyder det at der er mindre en 1Kb til rådighed for XML'en) er det nødvendigt at gemme det i flashen og indlæse det under afvikling af programmet.

Dette er gjort vha. PROGMEM[14] keywordet i Arduino, der instruerer kompileren om at variablen skal gemmes i flash og ikke i SRAM. Indlæsning af variablen under programafvikling fungerer som sædvanligt dog benyttes `strcpy_P` i stedet for `strcpy` (og tilsvarende for `strlen` og andre string-metoder). Derved opnås at det bliver muligt at gemme XML-beskrivelser, der fylder flash hukommelsen helt ud. Dette betyder at der er ca. 11Kb plads til XML'en, med denne chip.

Den anden ændring der er blevet indført, er at XML-beskrivelsen ikke længere sendes som en samlet blok. I stedet indlæses XML'en i mindre klumper (128 bytes af gangen i den nuværende udgave) som skrives på den serielle port sekventielt. Derved minimeres muligheden for at en stor klump XML overskriver andre variable i SRAM under indlæsning til den serielle port, samt at send-bufferen ikke bliver fyldt hvis modtageren ikke læser karaktererne hurtigt nok (send-bufferen er implementeret som en ringbuffer, der derfor nemt kan bliver korrupt hvis der skrives for meget data til bufferen).

6.1.4 Mobilplatformen

Implementeringen på mobilplatformen er bygget ovenpå den eksisterende MOURDA-platform, der giver mulighed for en event-dreven opbygning af interaktive historier. Udvidelserne til platformen er implementeret, så de følger konfigurations- og kommunikationsprotokollernes krav (afsnit 3.3 og 4.3.2). Dette har resulteret i en implementering, der opfylder de krav som er stillet i specialets formål og forventede resultater.

I dette delafsnit vil de problemstillinger og begrænsninger, der er fundet i forbindelse med test og verificering af implementeringen på mobilplatformen, blive diskuteret. I forbindelse med implementering og verificering af mobilplatformen er der også blevet identificeret en række mulige problemer, der vil blive analyseret herunder og eventuelle løsningsforslag til problemerne vil blive fremlagt.

Bluetooth-forbindelse

Telefonens fysiske Bluetooth-forbindelse kan være optaget, f.eks. af et Bluetooth-headset eller hvis det igangværende drama på telefonen bruger Bluetooth-forbindelsen til at sende/modtage data. Forbindelsen kan også være i brug sporadisk, af et andet program, og derved forhindre at mobilapplikationen kan benytte Bluetooth til at sende/modtage data konsekvent.

[14] http://www.arduino.cc/en/Reference/PROGMEM

I denne situation kan platformen ikke benytte Bluetooth-modulet til at søge efter og forbinde til de eksterne enheder, eller brugeren vil opleve at kommunikationen ikke fungerer som forventet (eller slet ikke). Det vil medføre en markant forringet brugeroplevelse af applikationen og eventuelt helt afholde en bruger fra at benytte den.

Ekstern brug af Bluetooth (f.eks. til headset eller andet der ikke kommer fra selve dramaet) er et i praksis umuligt problem at løse. Hvis Bluetooth ikke er tilgængeligt kan det ikke bruges. Men brugeren skal muligvis gøres opmærksom på dette i applikationen, forudsat at det er en tilstand som kan detekteres igennem Java ME-platformen.

Problemer med intern brug af Bluetooth (i samme applikation) kan afhjælpes ved at al Bluetooth-kommunikation pakkes ind i en fælles klasse, som er indeholdt i frameworket. Ved at have kommunikationen samlet ét sted, bliver det muligt at håndtere situationer hvor mere end én komponent ønsker at sende/modtage data.

Derudover er det et andet problem med Bluetooth-kommunikationen, i det tilfælde at en bruger starter en interaktiv komponent og efterfølgende bevæger sig væk fra den fysiske udstilling uden at afslutte komponenten. I denne situation vil brugeren opleve at applikationen stadig er aktiv, men ikke responsiv.

Håndtering af denne situation kan implementeres ved at det underliggende system automatisk undersøger om den aktuelle eksterne enhed er indenfor rækkevidde og ellers informere brugeren om at komponenten skal afsluttes. Dette vil samtidig give brugeren mulighed for at bevæge sig væk fra en udstilling midlertidigt og stadig have applikationen aktiv når der vendes tilbage. Det er også muligt at systemet automatisk kan lukke komponenten ned, hvis enheden er udenfor rækkevidde, da den alligevel vil blive tilgængelig igen når udstillingen er indenfor rækkevidde.

Igangværende drama

Den dynamiske udvidelse af historier, der er blevet implementeret, tager ikke højde for udformningen og indholdet af den igangværende historie. Dette er et bevidst valg, for at sikre generalitet i implementeringen og afviklingen af interaktive knuder. Ved at se bort fra det underliggende drama, er der mulighed for at de nye knuder kan indlæses, afvikles og lukkes ned uden at forstyrre det igangværende drama. Samtidig er der indbygget mulighed for at nye knuder kan kommunikere indirekte med det igangværende drama, enten ved at benytte modellen eller ved at sende/fange events.

Et problem der kan opstå i denne sammenhæng er at det underliggende drama stadig er aktivt i baggrunden. Hvis der sendes et event som fanges deri og der f.eks. vises en dialog eller der startes en anden aktivitet, som skifter fokus væk fra den interaktive knude, kan det resultere i at den interaktive knude ”forsvinder” og brugeren vender tilbage til det ”gamle” drama utilsigtet. Det underliggende drama kan også vise en modal dialog, der blokerer for input til den interaktive komponent, eller vise en ny komponent som skjuler den interaktive.

Disse problemer skyldes den meget afkoblede implementering af både MOURDA og udvidelsen dertil, hvor alt er baseret på events, og uafhængige komponenter/moduler. En løsning derpå kan være, at

interaktive knuder kan sætte en "lås" på systemet, f.eks. ved at sætte en værdi i modellen, eller ændre opførslen af de centrale NodeLoader og Eventbus-objekter, som er ansvarlige for håndtering af events og komponenter. Dette vil gøre det muligt for interaktive knuder at forblive synlige og altid have kontrolover det systemet.

Men det vil samtidig medføre at alle dramaer der implementeres skal tage højde for denne situation. Hvis et drama f.eks. har en timer der kører i baggrunden, skal denne også pauses mens den nye knude afvikles, for ikke at forstyrre flowet i det oprindelige drama. Løsningen kan eventuelt udvides så alle interaktive elementer i MOURDA (timere, events, tråde og lignende) implementeres som elementer i frameworket, der automatisk håndterer denne situation, hvis en bestemt værdi sættes i modellen. Hvis et nyt baggrundselement implementeres (f.eks. et modul der lytter på en asynkron besked via HTTP), skal denne designes så der tages højde for denne "systemlås".

6.1.5 Den samlede platform

Der er også blevet identificeret problemer i forbindelse med en samlede løsning. Disse problemer kræver ændringer/tilføjelser på begge sider af platformen (mobil og Arduino) og analysen af dem er mere generel og overordnet, da problemstillinger optræder i det grundlæggende design af systemet.

Flere/mange samtidige brugere

Et problem med den nuværende implementering, er at der ikke umiddelbart tages højde for hvordan systemet skal opføre sig hvis der er flere brugere, på samme interaktive knude, på samme tid.

Hvis knuden er en simpel historieknude, eller hvis den kun har sensorer, kan flere brugere benytte systemet samtidigt (dog med korte pauser når der skal læses data fra enheden). Men hvis knuden har indbygget interaktion, f.eks. ved at der er en række aktuatorer som brugeren kan påvirke, kan det give problemer med mange brugere.

Her kan man tænke sig et scenarie hvor en række brugere forsøger at ændre værdien på/tilstanden af en aktuator samtidigt og den indlejrede enhed vil derfor modtage en lang række forskellige kommandoer og afvikle dem allesammen i den modtagne rækkefølge. Det endelige resultat vil, for næsten alle brugerne, være forskelligt fra det valg de traf på mobiltelefonen. Samtidig kan det medføre at de fysiske komponenter, der er forbundet til den indlejrede enhed, får en masse kommandoer indenfor kort tid, som de måske ikke er i stand til at håndtere.

Problemet er her at der i protokollen ikke tages eksplicit højde for flere samtidige brugere. Det kan delvist afhjælpes ved at sætte en begrænsning på hvor ofte en indlejret enhed behøver afvikle en modtaget kommando, eller hvor ofte den behøver acceptere en forbindelse. Ved at sætte en tidsbegrænsning på kommandoerne, sikrer man at de fysiske komponenter ikke får for mange beskeder, samt at Arduino-enhederne ikke bliver blokeret af uhåndterbare kommandoer.

Men det kræver at der i protokollen indføres en mulighed for enten at Arduino-enheden kan sende en 'BUSY'-kommando tilbage eller at mobiltelefonen først skal spørge om lov, før den sender en kommando. Dette vil dog ikke løse alle problemerne da det blot vil nedsætte mængden af kommandoer, men ikke nødvendigvis vil give brugeren indtryk af at hun interagerer med den fysiske installation, i det tilfælde at en arbitrær pause indsættes i kommunikationen.

Brugergrænsefladen til aktuatorer skal måske også laves om, så tidsrummet uden mulighed for interaktion bliver en del af grænsefladen. Aktuatorværdier/-tilstande skal muligvis også opdateres automatisk, så brugeren altid kan se hvilken tilstand systemet er i, samt hvilke muligheder der er tilgængelige. Grænsefladen kan således fjerne kommandoer der ikke er tilgængelige ('tænd for lys' når lyset allerede er tændt) og derved minimere mængden af kommandoer der sendes til den indlejrede enhed.

Mængden af beskeder, der bliver sendt til den indlejrede enhed, vil selvfølgelig stige hvis værdien/tilstanden aflæses automatisk og hvis mobiltelefonerne først skal "spørge om lov" før de sender en kommando. Men problemet her er ikke så meget mængden af beskeder, men mere mængden af kommandoer der ændrer en værdi/tilstand på enheden.

Inden et system, baseret på dette framework, sættes i endelig drift, skal der foretages en ny test af brugeroplevelsen. Denne test forudsætter en implementering af de nævnte ændringer af protokollen, eller tilsvarende ændringer der afhjælper problemet med for mange samtidige kommandoer. Testen skal undersøge brugeroplevelsen af et interaktivt system, hvor interaktionen kan sættes på pause på vilkårlige tidspunkter. Det er ikke sikkert at den aktuelle grænsefladeimplementering er nok til at håndtere denne situation. Implementeringen af ændringerne og testen af brugeroplevelsen ligger udenfor rammerne af dette speciale.

Langsom Bluetooth-forbindelse / konstant forbindelse

Første implementering af kommunikationsprotokollen på Java ME-platformen benyttede sig af midlertidige Bluetooth-forbindelser der blev åbnet når der var brug for at sende/modtage data og lukket igen umiddelbart efter. Dette blev indført for at sikre mod at en mobilapplikation ikke blokerer den eksterne enheds Bluetooth-forbindelse i perioder hvor der ikke transmitteres data.

Dog viste test af hastigheden at denne metode var meget langsom og det gav en dårlig brugeroplevelse når der skulle ændres værdi/tilstand af en aktuator eller foretages aflæsning af en sensorværdi. Samtidig umuliggjorde det at sensorværdier automatisk blev opdateret hurtigere end worst-case hastigheden for oprettelsen af en forbindelse. Dette blev yderligere forværret i situationer hvor der var mere end én sensor der skulle opdateres.

Tabellen herunder viser målinger foretaget på mobiltelefonen. Tallene er i millisekunder og er regnet fra afsendelse af kommandoen påbegyndes til den returnerede værdi er blevet aflæst. Som tabellen viser, tager det i gennemsnit over tre sekunder at aflæse/ændre en værdi med denne implementering.

Måling	1	2	3	4	5	6	7	8	Gennemsnit
Aflæsning af sensor	3327	3196	3410	3367	3387	3319	3401	3154	3320
Ændring af aktuator	3954	4031	3831	3868	3378	3639	2969	3000	3584

Tabel 7: Tider for Bluetooth-kommunikation med oprettelse af forbindelse (i millisekunder)

Dette medfører en forringet brugeroplevelse og derfor er implementeringen blevet ændret til at forbindelsen holdes åben konstant (se afsnit 4.3.2 *Implementering i mobilapplikationen*). Tiderne for denne implementering kan ses i Tabel 8 herunder.

Måling	1	2	3	4	5	6	7	8	Gennemsnit
Aflæsning af sensor	531	682	636	492	489	467	610	624	566
Ændring af aktuator	500	711	425	458	401	457	909	540	550

Tabel 8: tider for Bluetooth-kommunikation med konstant åben forbindelse (i milllisekunder)

Designændringen har medført en hastighedsforøgelse på ca. faktor 6. Fra worst-case i den nye implementering til best-case i den oprindelige er der stadig en forbedring på mere end faktor 3.

Dog medfører denne ændring at de enkelte mobilapplikationer nu kan blokere den eksterne enheds Bluetooth-enhed og forhindre andre applikationer i at forbinde til enheden. Selvom den enkelte enhed ikke bliver afholdt fra at kunne forbinde, er protokollen for simpel til at håndtere denne situation.

Der kan eventuelt benyttes Bluetooth-enheder som kan være forbundet til flere klienter samtidigt og protokollen kan udvides med et pakkebegreb, så kommandoer pakkes ind i en struktur der indeholder afsender og kommando. Ligeledes med returværdier fra den eksterne enhed. Dette ville afhjælpe problemet med at flere enheder har en konstant forbindelse til Arduino-enheden.

6.2 Sammenligning med relateret arbejde

I afsnit 2.2.5 (Tabel 1) blev der opsat en metrik til sammenligning af relateret arbejde, der opdelte arbejdet i fem emner; interaktion, GUI-generering, service discovery, historieudvidelse og fysisk interaktion.

6.2.1 Sammenligningsmetrik

De fem emner er dette speciales hovedemner, og efter endt analyse, implementering og test er det nu muligt at indsætte specialets resultater i den samme metrik. Den samlede metriksammenligning kan ses nedenfor i Tabel 9. Specialet kan ses i den nederste række af tabellen.

Metrikken viser at der i specialet er blevet implementeret løsninger på alle hovedemnerne, samt at en lang række af de tilsvarende elementer, i de relaterede projekter, er blevet inddraget.

Celler markeret med rød (lilla) er de elementer, fra andre projekter, som berøres/benyttes i dette speciale. De vil blive diskuteret i det følgende afsnit.

Projekt	Interaktion	GUI-generering	Service discovery	Historieudvidelse	Fysiske enheder
ORIENT	Der opsættes et sæt af regler for design af interaktion.				
INSTEP	Interaktion håndteres igennem et kort, en række spil og multimedier. Brugeren sendes fysisk rundt i rummet.			Historien udvides igennem brugerens interaktion med udstillinger. Historien prædefineret men uden fastlagt kronologi.	Stationære opstillinger med tilkoblet audio/video. Der interageres vha. et udleveret kort.
PMIF	Via visuelle tags, RFID, NFC, Bluetooth eller andet interageres der med miljøet.		Visuelle tags, RFID, Bluetooth, mm.	Historien udvides igennem interaktion med miljøet.	Kommunikation igennem abstrakt PhysicalWorldConnector og et Java Stream-interface.
XForms	Post submit til en server/enhed via HTTP.	Højniveau-beskrivelser af abstrakt funktionalitet. Generiske form-beskrivelser.	Beskrivelse af services/enhe der sendes via HTTP-headere.		Fysiske enheder mappes til inputfelter via en headerbeskrivelse.
TERESA		Lagdelt task-model med abstrakte interactors og operatorer. Samme beskrivelse benyttes på multiple platforme.			
PUC	Abstrakte kommandoer angiver ønsket interaktion med den aktuelle enhed.	Abstrakt XML-beskrivelse vha. tilstande og kommandoer.			Kommunikation håndteres via adapterer til de enkelte enheder og en standard transmissions-protokol.
SPECIALE	Interaktion via Bluetooth. Fysiske enheder mappes til GUI-komponenter.	Abstrakt XML-beskrivelse, generelle komponenter og eventbeskrivelser.	Bluetooth-navne genkendes (mulighed for SDP)	Historien udvides dynamisk ved kommunikation med eksterne enheder.	Kommunikation via standard protokol over Bluetooth. Stream-interface benyttes til kommunikation.

Tabel 9: Sammenligningsmetrik med speciale

6.2.2 Diskussion af hovedemener

De forskellige hovedemner, og de elementer der er blevet inddraget fra de relaterede projekter, vil blive diskuteret herunder.

Interaktion

Interaktionen i *PUC*-projektet (se afsnit 2.1.8) minder meget om den, der er blevet implementeret i dette speciale. I *PUC* angives interaktion vha. abstrakte kommandoer der mappes til GUI-komponenter. Ved at manipulere disse komponenter i mobilens brugerflade, kan brugeren interagere med de eksterne enheder. Det er ikke fastlagt hvorledes interaktionen sættes i gang, denne del antages at blive implementeret senere.

I dette speciale benyttes en meget tilsvarende model, hvor de fysiske enheder mappes til GUI-komponenter og kommandoer kan sendes til den eksterne enhed ved at manipulere komponenterne.

I *PMIF*-projektet (se afsnit 2.1.4) består interaktionen af en række direkte interaktionsmuligheder, som f.eks. at scanne et visuelt tag eller berøre en NFC-sensor med telefonen. Det er ikke eksplicit interaktion i brugerfladen ud over denne interaktion.

Denne direkte interaktion er ikke umiddelbart understøttet i specialets protokol, men scanning af tags er understøttet i *MOURDA* (download af nye dramakomponenter vha. denne metode, er en relativ lille ændring af protokollen).

De retningslinjer der blev opstillet i *ORIENT*-projektet (se afsnit 2.1.2) er også blevet benyttet i dette speciales implementering.

Interaktionen med historien skal være transparent for brugeren.
Dette er opnået ved at gøre den fysiske interaktion med de eksterne enheder automatisk og ved at ”skjule” interaktionsobjekterne i den samme grafiske repræsentation som benyttes til historien.

Brugerens engagement kan øges ved at benytte velkendte interaktionsværktøjer.
I specialet benyttes mobiltelefoner til at opnå en velkendt brugergrænseflade og et velkendt interaktionsværktøj.

Typen af interaktion skal understøtte plottet/historien.
Det er muligt at vælge mange forskellige former for interaktioner med det udviklede framework og stadig benytte den samme mobile platform. Typen af interaktion kan derfor vælges så den understøtter historien.

GUI-generering

Dette afsnit antager at udvidelsen af konfigurationsprotokollen (fra afsnit 3.2.4 *Diskussion af udvidelser* af konfigurationsprotokol) er implementeret.

Ideen med at benytte sig af generelle højniveauprimitiver minder meget om den måde interfaces beskrives i nogle af de projekter, der er nævnt i afsnit 2.2.3 *Services og brugergrænsefladegenerering*. Mere specifikt svarer det til måden XForms benyttes i *’XForms*-projektet’ (se afsnit 2.1.7) og den måde hvorpå den abstrakte brugergrænseflade beskrives vha. interactors og operatorer i *TERESA*-projektet

(se afsnit 2.1.9). Her beskrives også kun en højniveaufunktionalitet og -interaktion og det er op til runtime-miljøet at tilpasse beskrivelsen til en konkret brugergrænseflade på den aktuelle enhed.

I XForms findes der en række abstrakte kontrolobjekter, som beskriver en ønsket funktionalitet (f.eks. select eller input), og ved at benytte sig af tilsvarende abstrakte beskrivelser kan den samme grad af afkoblethed og genbrugbarhed opnås i den konfigurationsprotokol, der er implementeret i dette speciale.

Brugen af abstrakte triggers og handlinger minder også om triggers fra *XForms* og operatorer fra *TERESA*-projektet. Det er ligeledes abstrakte højniveaubeskrivelser der benyttes til at angive en ønsket interaktion eller handling, frem for at binde beskrivelsen til implementeringen.

Selv om antagelsen fra starten af delafsnittet, om implementering af udvidelserne, ikke holder (som i prototypen), benyttes der stadig abstrakte primitiver til at beskrive interfaces og interaktion med. Dog ikke på samme høje niveau og med en hvis kobling til implementeringen.

Den implementerede protokol er stadig baseret på XML og den endelige implementering af beskrivelsen ligger ikke fast (på beskrivelsestidspunktet). Derudover er det muligt at benytte et væld af events, moduler og komponenter til at opnå meget komplicerede interaktioner, blot ved at lave en simpel beskrivelse af interaktionen med events og eventhandlers.

Service discovery

Service discovery er et emne som mange af de relaterede projekter har abstraheret væk og blot antaget at dette var implementeret (f.eks. *PUC*-projektet).

Det eneste projekt der også tog denne del med var *PMIF*-projeket hvor det endda ikke bliver entydigt defineret hvordan service skal findes. I *PMIF* benyttes service discovery til at finde services (kommunikationstypen) på enheder, der allerede er fundet.

I dette speciale er der blevet implementeret en simpel discovery protokol, samt fremlagt en mulighed for at udvide denne vha. en standardiseret protokol.

Historieudvidelse

Historieudvidelse blev hovedsageligt behandlet i *INSTEP*-projektet (se afsnit 2.1.5), men i *PMIF* blev denne del også indirekte behandlet.

I *PMIF* fungerede historieudvidelser ved at brugeren af applikationen henter en beskrivelse af faste services, miljøet eller andre brugere og viser denne i brugerfladen, så brugeren kan interagere med den.

Dette fungerer som en slags historieudvidelse da det er muligt at udvide det, brugeren er i gang med på mobiltelefonen, vha. denne teknologi. F.eks. kan applikationen benyttes til at læse en reklame for en film, købe en billet og printe den ud, blot ved at interagere med tre forskellige apparater.

I *INSTEP*-projektet blev brugeren ledt rundt på et museum ved at interagere med faste installationer. Historien blev afspillet som multimedier på disse installationer. Udvidelsen består i at brugerens valg og

færden afspejler sig i de medier, som afspilles til brugeren. Historien udvikler sig efterhånden som brugeren kommer længere frem og får besøgt flere installationer.

Udvidelserne i specialet minder meget om dem, der benyttes i *PMIF*-projektet. Begge dele er udvidelser af en igangværende historie igennem interaktion med en specifik ekstern enhed (definitionen af en historie er dog meget forskellig imellem de to projekter). *INSTEP*-projektets udvidelser er mere en praktisk ide til hvordan man kan implementere selve historieudvidelsen i et mobilt drama.

Fysiske enheder

Håndteringen af de fysiske enheder er et emne som mange af projekterne har omhandlet.

PMIF-projektet håndterer kommunikationen med de eksterne enheder ved at indkapsle disse i et abstrakt kommunikationsobjekt og ved at benytte højniveau Streams til den fysiske kommunikation. Og i *PUC*-projektet benyttes en række abstrakte adapterer, som har samme funktion.

Dette er meget lig den måde det er blevet implementeret på i dette speciale hvor der også benyttes et højniveau kommunikationsobjekt (der dog ikke er abstrakt på samme niveau men derimod koblet til Bluetooth-protokollen) og Stream-interfaces til at afkoble kommunikationen fra selve historieimplementeringen.

6.3 Fremtidigt arbejde

I dette afsnit vil det forventede fremtidige arbejde med frameworket blive diskuteret. Mulighederne for at benytte frameworket til at lave dynamiske interaktive historier vil ligeledes blive diskuteret.

6.3.1 Undersøgelser

Selvom frameworket er blevet implementeret i forhold til specialets formål og de opsatte krav, er der en række elementerne som skal undersøges før frameworket er klart til at blive benyttet i et færdigt system. Derudover er der flere muligheder for udvidelse af frameworket og dets design, som vil gøre det muligt at benytte i det i flere og mere generelle situationer.

Konfigurationsprotokollen

Konfigurationsprotokollen er implementeret så den svarer til de funktioner, der stiles til rådighed af de i prototypen benyttede enheder (sensorer og aktuatorer). Protokollen er ikke generisk nok til at kunne håndtere mange forskellige former for input og output, og de interaktionsmuligheder som gives af event-mekanismen er ligeledes bundet op på at det skal være simpel historieudvidelse.

Før protokollen kan benyttes i mere generelle applikationer, skal den derfor udvides så det bliver muligt at benytte mere generiske beskrivelser af funktionalitet og interaktion. Udvidelsen af konfigurationsprotokollen, som er beskrevet i afsnit 3.2.4 *Diskussion af udvidelser* af konfigurationsprotokol, skal undersøges og implementeres før frameworket opnår den nødvendige grad af generalitet. Denne protokolændring skal undersøges og sammenlignes med andre tilsvarende protokollen (som beskrevet i underafsnittet *GUI-generering* i afsnit 6.2.2), før den implementeres.

En udvidelse af konfigurationsprotokollen vil også medføre at der skal implementeres en række grafiske primitiver og komponenter, som kan benyttes til at visualisere de mange nye muligheder der tilføjes.

Der skal også laves nye events og en ny måde at beskrive interaktionen internt i en grafisk komponent. F.eks. er det på nuværende tidspunkt ikke muligt at sende et event fra et primitiv til et andet, når brugeren interagerer med brugerfladen. Denne ændring af måden events beskrives på, skal indføres i protokollen og de tilhørende events skal implementeres i frameworket.

Derudover skal interaktionskomponenterne implementeres så de kan afvikles i baggrunden af mobilapplikationen (altså uden en synlig brugerflade), ved at angive dette i XML-beskrivelsen. I den nuværende udgave er det kun Bluetooth-modulet der afvikles i baggrunden, men hvis frameworket udvides så interaktionen også kan afvikles i baggrunden, bliver det muligt at interagere med miljøet indirekte, altså blot ved at være i nærheden af en ekstern enhed, eller ved at et andet systemevent (en timer, afslutning af et lydklip eller andet) aktiverer interaktionen.

Tilsammen vil disse ændringer gøre frameworket tilpas generisk til at kunne beskrive brugergrænseflader til alle gængse typer af interaktion. Samtidig vil en ændring fra den implementeringsnære beskrivelse der benyttes nu, til en mere abstrakt funktionel beskrivelse gøre det muligt at benytte den samme konfigurationsprotokol til applikationer, der måske ikke benytter sig af MOURDA-frameworket eller som kun implementerer en delmængde deraf.

Kommunikationsprotokollen

Kommunikationsprotokollen er implementeret så den kan håndtere prototypens krav til kommunikation og understøtter de funktioner og interaktioner som konfigurationsprotokollen gør det muligt at beskrive. Denne tætte kobling til en praktisk implementering har medført at den simple protokol ikke nødvendigvis er i stand til at håndtere mere komplicerede eller avancerede systemer.

Håndtering af flere samtidige mobilapplikationer (på den samme indlejrede enhed) er ikke implementeret og protokollen sikrer ikke modtagelse af hverken kommandoer eller svar. Derfor skal kommunikationsprotokollen ændres, så det bliver muligt at benytte flere samtidige mobilapplikationer på en enkelt indlejret enhed (som beskrevet i afsnit 6.1.5 ovenfor).

Denne ændring er nødvendig for at sikre at systemet kan blive benyttet i mange forskellige typer af interaktive historier, hvor det netop kan være et kendetegn at der er flere samtidige brugere.

Kommunikationsprotokollen skal også gøres mere fejlsikker, da der på nuværende tidspunkt ikke er noget der sikrer at en besked når frem til den forventede modtager eller at der nogensinde kommer et svar tilbage. Dette medfører f.eks. at en mobilapplikation i den nuværende udgave vil vente uendeligt på et svar fra en enhed, som den har mistet den fysiske kontakt med, hvis der er afsendt en kommando men endnu ikke modtaget svar. Et timeout skal defineres i protokollen, så enheder ved hvor lang tid der skal gå, før en besked antages at være tabt.

Derudover kan der indføres et pakkebegreb i kommunikationen, så kommandoerne er indeholdt i en pakke som også indeholder afsender og modtager. Dette ville være med til at sikre at beskederne når frem til rette modtager, men det vil også medføre et stort overhead på kommunikationen. Specielt på

Arduino-siden kan dette blive et problem, da det er den mest begrænsede platform og samtidig den der skal håndtere flest beskeder.

Mere avancerede systemer kan medføre at de enkelte indlejrede enheder har mange sensorer/aktuatorer tilkoblet, hvilket medfører at de enkelte mobilapplikationer skal sende mange beskeder for at opdatere alle værdierne i brugergrænsefladen. En anden udvidelsesmulighed kan derfor være at tilføje en mulighed i protokollen for at spørge efter, eller sætte, flere værdier samtidigt. I stedet for at sende to kommander efter hinanden, og få to uafhængige svar tilbage, kan det gøres muligt for mobilapplikationen at spørge efter en række værdier og få dem alle returneret samtidigt. Det samme kan implementeres med skrivning af nye værdier/tilstande.

Dette vil mindske netværkstrafikken (med færre beskeder) og gøre det nemmere at understøtte systemer med mange fysiske enheder tilkoblet.

Service discovery

Service discovery protokollen er implementeret som en simpel navnesammenligning, men undersøgelsen viste at det er teoretisk muligt at benytte den indbyggede service discovery protokol i Bluetooth, til at lægge et ekstra lag ovenpå protokollen (se afsnit 4.5.1 *Analyse af service discovery protokollen*). Dette lag vil gøre det muligt at genkende de enheder som understøtter frameworket, uden at der behøver blive oprettet en seriel forbindelse først.

Ved at tilføje dette lag vil systemet blive mere stabilt (fordi mobilapplikationerne ikke behøver sikre sig imod ”falske” enheder med matchende navn) og samtidig gøre service discovery mere generelt på tværs af applikationer.

Derudover vil det være muligt at benytte forskellige UUID’er til at beskrive servicen med og derved opnå at der kan understøttes flere versioner af kommunikationsprotokoller, hvis dette er nødvendigt senere i frameworkets livscyklus. Mobilapplikationen kan så nøjes med at lede efter de serviceversioner som den understøtter.

Det samlede system

Udvidelser af konfigurationsprotokollen, og dermed ændringer af XML-beskrivelsens opbygning, vil medføre en fragmentering af protokollen i forhold til de implementerede systemer. En enhed der benytter sig af version X+1 til at beskrive sine funktioner, vil ikke nødvendigvis kunne blive vist af en mobilapplikation der kun understøtter version X. I værste tilfælde vil det medføre at mobilapplikationen går ned eller på anden vis kommer i en fejltilstand. En udvidelsesmulighed der vil være meget nyttigt på langt sigt, er derfor at tilføje et versionsnummer til XML-beskrivelsen og et versionsnummer til konfigurationsprotokollen.

Den første ændring er den simpleste og betyder at de indlejrede enheder skal holde styr på, hvilken version af XML-beskrivelsen der er aktuel (enheden behøver kun gemme den nyeste). Mobilapplikationen kan så spørge efter versionsnummeret, og hvis det er højere end det applikationen har hentet tidligere, kan den nye beskrivelse hentes ned. Derved bliver det muligt at opdatere de indlejrede enheder (f.eks. ved at skifte det fysiske hardware, der er tilkoblet, ud med noget andet) og få

opdateret beskrivelsen i mobilapplikationen, uden at mobilen konstant skal spørge efter XML-beskrivelsen.

Den anden ændring betyder at hver gang konfigurationsprotokollen laves om, laves der en ny beskrivelse som følger den nye protokol. Mobilapplikationen ved hvilke versioner af protokollen den understøtter og det bliver derfor muligt for mobilen kun at hente de beskrivelser, som er lavet med en understøttet protokol. Mobilapplikationen kan f.eks. spørge den eksterne enhed hvilken version af protokollen den benytter og kun hente beskrivelsen hvis versionen er understøttet. Eller enheden kan have flere beskrivelser liggende, i forskellige versioner, og mobilapplikationen kan hente en beskrivelse i en understøttet version.

6.3.2 Muligheder for dynamik og interaktion

Frameworket giver en række muligheder og stiller en række funktioner til rådighed, som gør det muligt at tilføje dynamiske udvidelser og fysisk interaktion til interaktive historier. Disse muligheder vil blive diskuteret herunder.

Dynamisk udvidelse

Frameworket kan i sin nuværende form benyttes til dynamisk udvidelse af alle former for interaktive historier, som er implementeret med frameworket. Historier beskrevet i det XML-format som benyttes i frameworket (baseret på *MOURDA*'s XML-beskrivelse) kan udvides automatisk blot ved at starte et Bluetooth-modul som lytter efter eksterne enheder. Historieudvidelserne vil derefter automatisk blive hentet når applikationen kommer forbi de indlejrede enheder.

Denne funktionalitet er implementeret så der ikke stilles krav til indholdet af beskrivelsen, hvorfor det vil være muligt at udvide alle typer af historier (baseret på dette XML-format).

Det vil f.eks. være muligt at håndtere rundvisningsapplikationer på denne måde, ved at applikationen på forhånd ikke indeholder den færdige historie. Det vil også være muligt at udvide en historie baseret på brugerens bevægelse og opførsel igennem de events og komponenter, der findes i frameworket.

Derudover kan historien, igennem de eksisterende events, afvikles på forskellige måder alt efter hvilke valg brugeren tidligere har foretaget sig. F.eks. kan der gemmes en variabel hvis brugeren har foretaget sig noget bestemt. En ny knude ser så om denne variabel er sat, hvis ikke sendes ét event, hvis den er sendes et andet. Det vil gøre det muligt at lave dynamiske historier hvor indholdet bestemmes af brugerens interaktion med det fysiske rum.

Det vil derfor i fremtiden være muligt at udvide interaktive historier baseret på brugeren valg og bevægelse, blot ved at benytte sig af dette framework.

Fysisk interaktion

Samtidig giver frameworket mulighed for at interagere med den fysiske verden, direkte fra historien, ved at benytte sig af de indlejrede enheder og den protokol der er opbygget.

Den fysiske kobling mellem mobilapplikationen og de eksterne enheder gør det muligt at lave interaktive historier hvor der er indbygget interaktion mellem historien, brugeren og det fysiske miljø.

Frameworket har mulighed for at mappe input og output på de indlejrede enheder direkte til grafiske komponenter på brugerfladen. Derved bliver det muligt for en bruger af systemet at påvirke fysiske enheder, som er koblet til disse input og output, blot ved at interagere med interfacet. Det er muligt at tilkoble vilkårligt mange forskellige enheder, så længe disse kan styres vha. de digitale outputs på Arduino-boardet. På samme måde er det muligt at aflæse mange forskellige former for værdier, så længe enhederne, hvor værdierne kommer fra, kan aflæses digitalt eller analogt fra boardet.

Det er samtidigt muligt at aktivere denne fysiske interaktion i baggrunden af mobilapplikationen (hvis dette implementeres) og derved interagere med miljøet uden at brugeren foretager sig bevidste valg i brugerfladen. F.eks. kan en dør åbnes når brugeren når til et bestemt sted i et lydklip eller en lampe kan tændes når brugeren er tæt på, uden at det kan ses i brugergrænsefladen.

På samme måde kan en interaktion med miljøet forsage en ændring i brugerfladen, ved at mobilapplikationen lytter på en tilstand fra den eksterne enhed (som er forbundet til det fysiske element, som brugeren kan interagere med). En ændring ved elementet (tryk på en knap, stå et bestemt sted på gulvet, bevægelse i rummet, osv.) kan så aktivere en historieknude på telefonen.

Kapitel 7. Konklusion

I dette kapitel opsummeres resultaterne af specialet og de sammenlignes med det opsatte formål og de forventede resultater. Indholdet af specialet bliver kort opsummeret og analyserne og implementeringen af den endelige løsning bliver ligeledes præsenteret.

7.1 Kort opsummering

Specialet omhandler undersøgelsen af en mulighed for at understøtte dynamisk udvidelse af interaktive historier, afviklet på mobiltelefoner, ved at koble disse trådløst sammen med indlejrede enheder.

De indlejrede enheder indeholder en beskrivelse af et eller flere historieelementer, som hentes af mobiltelefonen og afvikles på lige fod med den igangværende historie.

Derudover er det muligt at interagere med det fysiske miljø omkring historien, ved at de indlejrede enheder er forbundet til fysiske elementer, som visualiseres igennem historiebeskrivelsen og kan påvirkes af brugeren. Brugergrænsefladen til at foretage denne interaktion genereres automatisk af mobilapplikationen.

I specialet er en række relaterede projekter blevet undersøgt og deres resultater er blevet sammenlignet med specialets formål og resultater (Kapitel 2).

Baggrundsundersøgelsen viste at der er mange forskellige forskningsprojekter der arbejder med dette emne, om end ingen af dem med held tidligere har kombineret den dynamiske historieudvidelse med fysisk interaktion.

Ikke at alle projekterne havde dette formål, en del projekter omhandlede dynamisk udvidelse af historier eller konfiguration af historier mens de afvikles. I disse projekter blev en række problemstillinger identificeret, hvor løsningen kunne være brugen af indlejrede enheder og trådløs konfiguration af nye historieelementer.

Analysen af de relaterede projekter viste samtidig endnu tydeligere at der er mange projekter der har forsøgt at implementere dynamiske interaktive historier, de fleste med held, men stadig i begrænset omfang. Andre projekter beskæftigede sig kun med automatisk konfiguration af brugergrænseflader, i forhold til funktionalitet stillet til rådighed af en ekstern enhed. Her er der ligeledes blevet udviklet en række frameworks, og her var antagelsen oftest at det omkringliggende (f.eks. historien) kommer andetsteds fra. Eller at enhederne kender hinanden på forhånd.

Herefter blev en konfigurationsprotokol analyseret og implementeret som en udvidelse af et eksisterende historieframework (*MOURDA*). Konfigurationsprotokollen benyttes til at generere en brugergrænseflade, der svarer til de funktioner som en ekstern enhed stiller til rådighed, eller til at udvide historien (Kapitel 3).

Der er også blevet udviklet en kommunikationsprotokol til kommunikation mellem mobilapplikationen og de indlejrede enheder. Protokollen er simpel og er koblet sammen med konfigurationsbeskrivelsen, for at

interaktionsmulighederne i brugergrænsefladen kan blive genereret automatisk af mobilapplikationen (Kapitel 4).

Endeligt er der blevet implementeret en prototype af frameworket, som demonstrerer mulighederne for at udvide en historie og interagere med fysiske installationer, der er forbundet til de indlejrede enheder (Kapitel 5).

Til sidst er der blevet foretaget en evaluering af de opnåede resultater, reflekteret over det relaterede arbejde og diskuteret hvilket fremtidigt arbejde der er nødvendigt (Kapitel 6).

7.2 Opnåelse af formål

Specialets formål var at undersøge følgende spørgsmål:
Hvordan understøttes interaktive historier med dynamisk udvidelse, igennem fysisk interaktion med indlejrede enheder?

Dette spørgsmål skulle undersøges ved at besvare fem delspørgsmål, som er beskrevet i afsnit 1.2 *Formål*.

Derudover var der opstillet en række forventede resultater i afsnit 1.6.

Forventede resultater

De forventede resultater er blevet opnået igennem de analyser og den implementering der er blevet foretaget i løbet af dette speciale.

Der er blevet implementeret et generisk softwareframework, der giver mulighed for at udvide en interaktiv historie mens den afvikles og mulighed for at skabe fysisk interaktion, ved at kommunikere med eksterne enheder (1.6 A). Det muliggør at eksterne enheder kan sende en beskrivelse til mobilapplikationen, som automatisk generer en brugergrænseflade der matcher enhedens funktioner, eller som beskriver en udvidelse af historien. Igennem denne brugerflade kan brugeren så interagere med den eksterne enhed. Frameworket er blevet implementeret både på historieplatformen og på en indlejret platform til de eksterne enheder.

Derudover er der blevet implementeret en proof-of-concept prototype, ved at benytte dette framework, som er i stand til at udvide en igangværende historie og interagere med fysiske enheder (1.6 B). Prototypen demonstrerede hvordan en interaktiv historie kan udvides dynamisk og hvordan fysik interaktion med det omgivende miljø kan opnås igennem kommunikation med de indlejrede enheder.

Endeligt er frameworkets tilstand og fremtidsmuligheder blevet diskuteret, i forhold til udvidelser af interaktive historier og i forhold til en forventet nødvendig implementering (1.6 C). I diskussionen blev en række problemstillinger ved den nuværende implementering identificeret og mulige løsningsforslag dertil blev opstillet.

Delspørgsmål

Igennem opnåelsen af disse resultater er de fem delspørgsmål fra formålet også blevet besvaret.

Delspørgsmål I og II er blevet besvaret under undersøgelsen og implementeringen af konfigurationsprotokollen og udvidelsen af MOURDA-frameworket i Kapitel 3. Her er det blevet vist hvordan indlejrede sensorer og aktuatorer kan vises grafisk på en mobiltelefon og hvordan denne repræsentation kan skabes dynamisk uden at enhederne kender hinanden på forhånd.

Delspørgsmål III og IV er blevet besvaret under undersøgelsen og implementeringen af kommunikationsprotokollen, service discovery-protokollen og Arduino-frameworket i Kapitel 4. Her det blevet vist hvordan fysisk interaktion med indlejrede enheder håndteres fra en mobiltelefon og hvordan denne interaktion skabes gennem et kendt interface til den indlejrede enhed.

Delspørgsmål V er blevet besvaret igennem diskussionen af udvidelsesmuligheder for konfigurationsprotokollen i afsnit 6.1.2. Her blev der diskuteret forskellige muligheder for at udvide protokollen og gøre den mere generisk, så den kan håndtere flere forskellige interaktionsformer og mere avanceret dynamisk historiefortælling.

Formål

Besvarelsen af de fem delspørgsmål i formålet har været med til at behandle og undersøge den problemstilling, som formålet søgte afklaret.

Igennem analyser af de forskellige elementer i formålet er problemstillingerne blevet belyst og igennem implementering af en prototype er det blevet vist hvordan resultatet implementeres i praksis. Resultaterne opnået i de forskellige afsnit, og i implementeringen af prototypen, har vist hvordan dynamisk udvidelse af interaktive historier kan understøttes af fysisk interaktion med indlejrede enheder, hvilket besvarer hovedspørgsmålet fra formålet.

7.3 Konklusion på projektet

Formålet med specialet var at undersøge hvordan man understøtter en dynamisk udvidelse af mobile interaktive historier, ved at interagere med den fysiske verden igennem indlejrede enheder. At finde en samlet løsning på de opstillede problemstillinger, hvor de dynamiske historier og interaktionen med eksterne enheder kombineres i ét framework.

Igennem specialets undersøgelser blev der fundet løsninger på nogle af de problemstillinger der var blevet opridset i afsnit 1.1 *Baggrund*, og disse løsninger blev realiseret i et softwareframework, der kan bruges til at lave dynamiske interaktive historier.

Det endelige framework består af en række protokoller (en konfigurations-, en kommunikations- og en service discovery-protokol), som tilsammen gør det muligt at udvide interaktive historier og interagere med indlejrede enheder. Dette er blevet implementeret som to softwareframeworks; et Java ME-framework til mobiltelefoner (hvor selve historien afvikles) og et Arduino-framework til de indlejrede enheder der gør det muligt at interagere med tilkoblede fysiske enheder.

Konfigurationsprotokollen er en XML-baseret interfacebeskrivelse, der beskriver indholdet af en brugergrænseflade ved hjælp af en række generelle interfacekomponenter. XML-beskrivelsen benyttes af mobilapplikationen til automatisk at genere en brugergrænseflade når beskrivelsen bliver hentet fra en ekstern enhed. Beskrivelsen er en abstraktion af den ønskede grænseflades funktionalitet og den er opbygget af abstrakte UI-komponenter og en interaktionsbeskrivelse bestående af sensorer/aktuatorer og events.

Kommunikationsprotokollen er en simpel tekst-baseret protokol som gør det muligt for mobilapplikationen og de eksterne enheder at kommunikere med hinanden. Igennem denne protokol kan mobilapplikationen hente en XML-beskrivelse af en ekstern enhed (som så kan bruges til at generere en brugergrænseflade med) eller aflæse/sætte værdier på de fysiske enheder der er tilkoblet Arduino-boardet.

Service discovery-protokollen gør det muligt for mobilapplikationen at finde de eksterne enheder automatisk, ved at sammenligne deres synlige navne med et kendt præfiks som er applikationsspecifikt. Kun enheder med det rette navnepræfiks bliver benyttet af applikationen.

Det endelige framework blev sammenholdt med kravene, formålet og det relaterede arbejdes resultater for at sikre at de oprindelige problemstillinger var blevet belyst og afhjulpet med denne implementering og det blev fundet at løsningen besvarer det opstillede hovedspørgsmål og de relaterede delspørgsmål. Den har ligeledes afhjulpet de problemstillinger som blev identificeret i det relaterede arbejde og inkorporeret de resultater som var blevet fundet deri.

Den implementerede prototype, som benytter sig af de implementerede softwareframeworks og protokoller, viste at de enkelte implementeringer opfylder de opstillede krav. Prototypen demonstrerede både historieudvidelse og interaktion med det fysiske miljø som historien afvikles i (igennem kommunikation med indlejrede enheder). Samtidig viste prototypen at der er en række elementer i protokollerne som skal undersøges nærmere eller udvides med forbedringer, før de to frameworks kan benyttes i kommercielle implementeringer af dynamiske interaktive historier. En række løsningsforslag til disse problemer er blevet opstillet i specialet.

Fremadrettet vil dette framework derfor kunne understøtte dynamisk udvidelse af interaktive historier, igennem fysisk interaktion med indlejrede enheder.

Det kan derfor konkluderes at formålet med specialet er blevet opnået.

Kapitel 8. Litteratur

Alle referencer i denne liste kan findes på den vedlagte CD i mappen `/Referencer` eller på nettet på http://speciale.barmonger.org/referencer.php

Barton, J., Kindberg, T., Dai, H. & Priyantha, N.B. 2003, "Sensor-enhanced mobile web clients: an XForms approach", *Proceedings of the 12th international conference on World Wide Web*ACM, , pp. 89.

Bichard, J., Brunnberg, L., Combetto, M., Gustafsson, A. & Juhlin, O. 2006, "Backseat Playgrounds: Pervasive Storytelling in Vast Location Based Games", *Lecture Notes in Computer Science,* vol. 4161, pp. 117.

Broll, G., Siorpaes, S., Rukzio, E., Paolucci, M., Hamard, J., Wagner, M. & Schmidt, A. 2007, "Supporting mobile service usage through physical mobile interaction", *5th Annual IEEE International Conference on Pervasive Computing and Communications, White Plains, NY, USA*Citeseer, .

Cai, Y., Miao, C., Tan, A.H. & Shen, Z. 2006, "Fuzzy cognitive goal net for interactive storytelling plot design", *Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*ACM, , pp. 56.

Danks, M., Goodchild, M., Rodriguez-Echavarria, K., Arnold, D.B. & Griffiths, R. 2007, "Interactive storytelling and gaming environments for museums: The interactive storytelling exhibition project", *LECTURE NOTES IN COMPUTER SCIENCE,* vol. 4469, pp. 104.

Flintham, M., Giannachi, G., Benford, S. & Adams, M. 2007, "Day of the Figurines: supporting episodic storytelling on mobile phones", *LECTURE NOTES IN COMPUTER SCIENCE,* vol. 4871, pp. 167.

Gajos, K. & Weld, D.S. 2004, "SUPPLE: automatically generating user interfaces", *Proceedings of the 9th international conference on Intelligent user interfaces*ACM New York, NY, USA, , pp. 93.

Gustafsson, A., Bichard, J., Brunnberg, L., Juhlin, O. & Combetto, M. 2006, "Believable environments: generating interactive storytelling in vast location-based pervasive games", *Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*ACM, , pp. 24.

Hansen, F.A., Kortbek, K.J. & Grønbæk, K. 2008, "Mobile Urban Drama-Setting the Stage with Location Based Technologies", *Proc. of 1st Joint Int. Conf. on Interactive Digital Storytelling*Springer, , pp. 26.

Hornecker, E. & Stifter, M. 2006, "Learning from interactive museum installations about interaction design for public settings", *Proceedings of the 18th Australia conference on Computer-Human Interaction: Design: Activities, Artefacts and Environments*ACM, , pp. 142.

Kortbek, K.J. & Grønbæk, K. 2008, "Interactive spatial multimedia for communication of art in the physical museum space", *Proceeding of the 16th ACM international conference on Multimedia*ACM, , pp. 609.

Kurdyukova, E., André, E. & Leichtenstern, K. 2009, "Introducing Multiple Interaction Devices to Interactive Storytelling: Experiences from Practice", *International Conference on Interactive Digital Storytelling, ICIDS, Guimarães; Springer Verlag (this volume)*Springer, .

Leichtenstern, K., André, E. & Vogt, T. 2007, "Role assignment via physical mobile interaction techniques in mobile multi-user applications for children", *Lecture Notes in Computer Science,* vol. 4794, pp. 38.

Merabti, M., Rhalibi, A.E., Shen, Y., Daniel, J., Melendez, A. & Price, M. 2008, "Interactive Storytelling: Approaches and Techniques to Achieve Dynamic Stories", *Lecture Notes in Computer Science,* vol. 5080, pp. 118-134.

Mori, G., Paterno, F. & Santoro, C. 2004, "Design and development of multidevice user interfaces through multiple logical descriptions", *IEEE Transactions on Software Engineering,* vol. 30, no. 8, pp. 507-520.

Nichols, J., Myers, B.A., Higgins, M., Hughes, J., Harris, T.K., Rosenfeld, R. & Pignol, M. 2002, "Generating remote control interfaces for complex appliances", *Proceedings of the 15th annual ACM symposium on User interface software and technology*ACM New York, NY, USA, , pp. 161.

Paay, J., Kjeldskov, J., Christensen, A., Ibsen, A., Jensen, D., Nielsen, G. & Vutborg, R. 2008, "Location-based storytelling in the urban environment", *Proceedings of the 20th Australasian Conference on Computer-Human Interaction: Designing for Habitus and Habitat*ACM, , pp. 122.

Reitmaier, T. & Marsden, G. 2009, "Bringing Digital Storytelling to the Mobile", *Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction: Part I*Springer, , pp. 753.

Riedl, M.O. & Stern, A. 2006, "Believable agents and intelligent story adaptation for interactive storytelling", *Lecture Notes in Computer Science,* vol. 4326, pp. 1.

Rukzio, E., Wetzstein, S. & Schmidt, A. 2005, "A Framework for Mobile Interactions with the Physical World", *Proceedings of Wireless Personal Multimedia Communication (WPMC'05),* .

Scheible, J., Tuulos, V.H. & Ojala, T. 2007, "Story Mashup: Design and evaluation of novel interactive storytelling game for mobile and web users", *Proceedings of the 6th international conference on Mobile and ubiquitous multimedia*ACM, , pp. 139.

Kapitel 9. Appendiks

De fleste elementer i appendikset kan også findes på den vedlagte CD eller på nettet på adressen http://speciale.barmonger.org

9.1 Prototypen

Alle filer der hører til den færdige prototype, er at finde på den vedlagte CD i mappen `/Prototype` eller på nettet http://speciale.barmonger.org/prototype.php

9.1.1 Applikation

En kompileret udgave af prototypeprogrammet (til Java ME-kompatible telefoner) findes i undermappen `/Applikation`.

Programmet findes i to versioner:

- En standardversion (der benytter sig af de fysiske Arduino-enheder)
- En demoversion (der benytter sig af MockDevice til at simulere de eksterne enheder).

Programmet er testet på en SonyEricsson W910-mobil.

9.1.2 Billeder

I undermappen `/Billeder` findes der en række billeder af de forskellige udstillinger, samt billeder af mobilapplikationen der interagerer med dem.

9.1.3 Video

I undermappen `/Video` er det muligt at finde optagelser af der samlede system i brug. Videoen demonstrerer hvordan applikationen fungerer i praksis og hvordan systemet benyttes.

9.2 Konfigurationsprotokol

Konfigurationsprotokollen er beskrevet i afsnit 3.2 *Konfigurationsprotokol*.

9.2.1 XML-eksempler

Herunder vises en række eksempler på hvordan den endelige konfigurationsprotokol kan benyttes i praksis. Eksemplerne er taget fra prototypeapplikationen (se afsnit 5.1 *Historie*) og XML-beskrivelserne herunder er de beskrivelser, der blev benyttet deri.

XML-filerne kan også findes på den vedlagte CD i mappen `/XML` eller på nettet på http://speciale.barmonger.org/xml.php.

Van Gogh

XML-beskrivelsen herunder beskriver den eksterne enhed, der viser flere billeder af Van Gogh og information der knytter sig til billederne.

```
<?xml version="1.0" encoding="UTF-8"?>
<interactive>

  <properties>
    <property key='discoveryQuestion'> Vil du se flere vaerker af Van Gogh?</property>
  </properties>

  <eventhandlers>
    <handler event='net.interactivespaces.mud.events.OnLoadEventHandler'>
      <guard>
        <properties>
          <property key='filename'>VanGogh.xml</property>
        </properties>
      </guard>
      <action name='net.interactivespaces.mud.actions.LoadGlobalResourcesAction'>
        <properties>
          <property key='resources'>
            <property key='1'>gogh1</property>
            <property key='2'>gogh2</property>
            <property key='3'>gogh3</property>
          </property>
        </properties>
      </action>
      <action name='net.interactivespaces.mud.actions.GotoNodeAction'>
        <properties>
          <property key='node'>vangogh1</property>
        </properties>
      </action>
    </handler>
  </eventhandlers>

  <resources baseUrl='http://barmonger.org/speciale'>
    <resource type='image/png' sizeInBytes='13519' source='' target='' filename='gogh1'
handle='gogh1'/>
    <resource type='image/png' sizeInBytes='13511' source='' target='' filename='gogh2'
handle='gogh2'/>
    <resource type='image/png' sizeInBytes='13519' source='' target='' filename='gogh3'
handle='gogh3'/>  </resources>

  <nodes>
    <node id='vangogh1' component='net.interactivespaces.mud.components.PictureAndText'>
      <properties>
        <property key='textkey'> Et maleri af en gadecafe om aftenen (The Cafe Terrace on the
Place du Forum) malet af Van Gogh i 1888 </property>
        <property key='imagekey'> gogh1 </property>
        <property key='style' ref='pictureAndTextStyle'/>
      </properties>
      <modules>
```

```
        <module id='backknap' type='net.interactivespaces.mud.modules.ButtonModule'>
          <properties>
            <property key='buttontextkey'> builtin.cmd.next </property>
          </properties>
        </module>
      </modules>
      <eventhandlers>
        <handler event='net.interactivespaces.mud.events.ButtonPressedHandler'>
          <action name='net.interactivespaces.mud.actions.GotoNodeAction'>
            <properties>
              <property key='node'>vangogh2</property>
            </properties>
          </action>
        </handler>
      </eventhandlers>
    </node>

    <node id='vangogh2' component='net.interactivespaces.mud.components.PictureAndText'>
      <properties>
        <property key='textkey'> Et oliemaleri af en vej med en cypres (Road with Cypress and
Star) malet af Van Gogh i 1890 </property>
        <property key='imagekey'> gogh2 </property>
        <property key='style' ref='pictureAndTextStyle'/>
      </properties>
      <modules>
        <module id='backknap' type='net.interactivespaces.mud.modules.ButtonModule'>
          <properties>
            <property key='buttontextkey'> builtin.cmd.next </property>
          </properties>
        </module>
      </modules>
      <eventhandlers>
        <handler event='net.interactivespaces.mud.events.ButtonPressedHandler'>
          <action name='net.interactivespaces.mud.actions.GotoNodeAction'>
            <properties>
              <property key='node'>vangogh3</property>
            </properties>
          </action>
        </handler>
      </eventhandlers>
    </node>

    <node id='vangogh3' component='net.interactivespaces.mud.components.PictureAndText'>
      <properties>
        <property key='textkey'> Et selvportraet af Van Gogh med en straahat</property>
        <property key='imagekey'> gogh3 </property>
        <property key='style' ref='pictureAndTextStyle'/>
      </properties>
      <modules>
        <module id='backknap' type='net.interactivespaces.mud.modules.ButtonModule'>
          <properties>
            <property key='buttontextkey'> builtin.cmd.done </property>
          </properties>
        </module>
      </modules>
      <eventhandlers>
        <handler event='net.interactivespaces.mud.events.ButtonPressedHandler'>
          <action name='net.interactivespaces.mud.actions.ReturnToMenuAction'>
          </action>
        </handler>
      </eventhandlers>
    </node>
  </nodes>

</interactive>
```

Eksempel 32: XML-beskrivelse til Van Gogh

Rumsensor

Den næste XML-beskrivelse beskriver den interaktive knude, der stiller temperatur- og lyssensor til rådighed.

```
<?xml version="1.0" encoding="UTF-8"?>
<interactive>

  <eventhandlers>
    <handler event='net.interactivespaces.mud.events.OnLoadEventHandler'>
      <guard>
        <properties>
          <property key='filename'>Sensor.xml</property>
        </properties>
      </guard>
      <action name='net.interactivespaces.mud.actions.GotoNodeAction'>
        <properties>
          <property key='node'>sensor</property>
        </properties>
      </action>
    </handler>
  </eventhandlers>

  <nodes>
    <node id='sensor' component='net.interactivespaces.mud.components.Dynamic'>
      <properties>
        <property key='deviceName'>Sensor</property>
        <property key='textKey'> Her kan du aflaese temperaturen i rummet og hvor meget lys der
er. </property>
        <property key='styleKey'>dynamicStyle</property>
        <property key='title'>Rumsensorer</property>
        <property key='sensors'>
          <property key='light'>
            <property key='name'> light </property>
            <property key='display'> Lys </property>
            <property key='type'> Horizontal </property>
            <property key='update'> auto </property>
            <property key='interval'> 5 </property>
            <property key='unit'> pct </property>
            <property key='max'> 100 </property>
            <property key='min'> 0 </property>
          </property>
          <property key='temperature'>
            <property key='name'> temp </property>
            <property key='display'> Temperatur </property>
            <property key='type'> Horizontal </property>
            <property key='update'> auto </property>
            <property key='interval'> 5 </property>
            <property key='unit'> C </property>
            <property key='max'> 45 </property>
            <property key='min'> 0 </property>
          </property>
        </property>
      </properties>
      <modules>
        <module id='back' type='net.interactivespaces.mud.modules.ButtonModule'>
          <properties>
            <property key='buttontextkey'> builtin.cmd.done </property>
          </properties>
        </module>
      </modules>
      <eventhandlers>
        <handler event='net.interactivespaces.mud.events.ButtonPressedHandler'>
          <action name='net.interactivespaces.mud.actions.ReturnToMenuAction'>
          </action>
        </handler>
      </eventhandlers>
    </node>
  </nodes>
</interactive>
```

Eksempel 33: XML-beskrivelse af rumsensorer

MiniKvarium

Den næste XML-beskrivelse beskriver den interaktive knude, der benyttes til at styre lys og bevægelse i MiniKvariet.

```
<?xml version="1.0" encoding="UTF-8"?>
<interactive>

  <eventhandlers>
    <handler event='net.interactivespaces.mud.events.OnLoadEventHandler'>
      <guard>
        <properties>
          <property key='filename'>MiniQuarium.xml</property>
        </properties>
      </guard>
      <action name='net.interactivespaces.mud.actions.LoadGlobalResourcesAction'>
        <properties>
          <property key='resources'>
            <property key='1'>aqua</property>
          </property>
        </properties>
      </action>
      <action name='net.interactivespaces.mud.actions.GotoNodeAction'>
        <properties>
          <property key='node'>aqua1</property>
        </properties>
      </action>
    </handler>
  </eventhandlers>

  <resources baseUrl='http://barmonger.org/speciale'>
    <resource type='image/png' sizeInBytes='54697' source='' target='' filename='aqua'
handle='aqua'/>
  </resources>

  <nodes>
    <node id='aqua1' component='net.interactivespaces.mud.components.PictureAndText'>
      <properties>
        <property key='textkey'> MiniKvariet er en interaktiv udstilling, hvor du kan se to
gopler der lever i naturlige omgivelser. </property>
        <property key='imagekey'> aqua </property>
        <property key='style' ref='pictureAndTextStyle'/>
      </properties>
      <modules>
        <module id='backknap' type='net.interactivespaces.mud.modules.ButtonModule'>
          <properties>
            <property key='buttontextkey'> builtin.cmd.next </property>
          </properties>
        </module>
      </modules>
      <eventhandlers>
        <handler event='net.interactivespaces.mud.events.ButtonPressedHandler'>
          <action name='net.interactivespaces.mud.actions.GotoNodeAction'>
            <properties>
              <property key='node'>aqua2</property>
            </properties>
          </action>
        </handler>
      </eventhandlers>
    </node>

    <node id='aqua2' component='net.interactivespaces.mud.components.Dynamic'>
      <properties>
        <property key='deviceName'>MiniQuarium</property>
        <property key='textKey'> Du kan styre MiniKvariet i menuerne herunder. </property>
        <property key='styleKey'>dynamicStyle</property>
        <property key='title'>MiniKvarium</property>
        <property key='actuators'>
          <property key='actuatorX'>
            <property key='type'> state </property>
            <property key='name'> pump </property>
            <property key='display'> Bevaegelse </property>
```

```
            <property key='states'>
              <property key='1'>Start</property>
              <property key='2'>Stop</property>
            </property>
          </property>
          <property key='actuatorY'>
            <property key='type'> state </property>
            <property key='name'> light </property>
            <property key='display'> Lys </property>
            <property key='states'>
              <property key='1'>Taend</property>
              <property key='2'>Sluk</property>
            </property>
          </property>
        </property>
      </properties>
      <modules>
        <module id='back' type='net.interactivespaces.mud.modules.ButtonModule'>
          <properties>
            <property key='buttontextkey'> builtin.cmd.done </property>
          </properties>
        </module>
      </modules>
      <eventhandlers>
        <handler event='net.interactivespaces.mud.events.ButtonPressedHandler'>
          <action name='net.interactivespaces.mud.actions.ReturnToMenuAction'>
          </action>
        </handler>
      </eventhandlers>
    </node>
  </nodes>
</interactive>
```

Eksempel 34: XML-beskrivelse af MiniKvariet

Museumsdrama

XML-beskrivelsen herunder er den, der benyttes i mobilapplikationen til at beskrive selve museumsdramaet. Properties er blevet fjernet fra dramaet i eksemplet, da elementet er meget langt og uoverskueligt og ikke bidrager med oplysninger omkring opbygningen af dramaet.

```
<?xml version="1.0" encoding="utf-8"?>
<drama>
  <meta>
    <id>specialetest</id>
    <name>Speciale</name>
  </meta>

  <properties>
    <!-- Fjernet pga. overskuelighed -->
  </properties>

  <versions>
    <version startNode="mainmenu" languageHandle="language">Dansk</version>
  </versions>

  <resources baseUrl="http://barmonger.org/speciale">
    <resourceSet id="dk">
        </resourceSet>
  </resources>

  <global>

    <eventhandlers>
      <handler event="net.interactivespaces.mud.events.BTSearchCompleteHandler">
        <action name="net.interactivespaces.mud.actions.ShowInformationAction">
          <properties>
            <property key="message">mus.guide.msg</property>
            <property key="title">mus.guide.title</property>
            <property key="imageHandle">guide</property>
          </properties>
        </action>
        <action name="net.interactivespaces.mud.actions.ChangeModulePropertiesAction">
          <properties>
            <property key="moduleId">bt_module</property>
            <property key="search">true</property>
          </properties>
        </action>>
      </handler>

      <handler event="net.interactivespaces.mud.events.DevicesInRangeHandler">
        <action name="net.interactivespaces.mud.actions.HandleDevicesAction">
          <properties>
            <property key="deviceChoiceId">showDevices</property>
          </properties>
        </action>
      </handler>
    </eventhandlers>

    <nodes>
      <node id="showDevices" component="net.interactivespaces.mud.components.DeviceChoice">
        <properties>
          <property key="titleKey">mus.exhibits.title</property>
          <property key="textKey">mus.exhibits.msg</property>
          <property key="image">guide</property>
          <property key="style" ref="deviceChoiceStyle"/>
        </properties>
        <modules/>
        <eventhandlers>
          <handler event="net.interactivespaces.mud.events.ComponentStartedHandler">
            <action name="net.interactivespaces.mud.actions.ChangeModulePropertiesAction">
              <properties>
                <property key="moduleId">bt_module</property>
                <property key="search">true</property>
              </properties>
            </action>
          </handler>
```

```
      </eventhandlers>
    </node>
  </nodes>

  <modules>
    <module id="bt_module" type="net.interactivespaces.mud.modules.BluetoothModule">
      <properties>
        <property key="search">false</property>
      </properties>
    </module>
  </modules>

</global>

<nodes>
  <node id="mainmenu" component="net.interactivespaces.mud.components.Choice">
    <properties>
      <property key="question">mainmenu.question</property>
      <property key="choices">
        <property key="1">mainmenu.choice1</property>
        <property key="2">mainmenu.choice2</property>
      </property>
      <property key="style" ref="choiceStyle"/>
    </properties>
    <modules/>

    <eventhandlers>
      <handler event="net.interactivespaces.mud.events.ComponentStartedHandler">
        <action name="net.interactivespaces.mud.actions.IfAction">
          <properties>
            <property key="function">isSetInModel</property>
            <property key="modelKey">bt_searching</property>
          </properties>
        </action>
      </handler>
      <handler event="net.interactivespaces.mud.events.IfActionResponseHandler">
        <guard>
          <properties>
            <property key="result"> yes </property>
          </properties>
        </guard>
        <action name="net.interactivespaces.mud.actions.ShowInformationAction">
          <properties>
            <property key="message">mus.guide.msg</property>
            <property key="title">mus.guide.title</property>
            <property key="imageHandle">guide</property>
          </properties>
        </action>
        <action name="net.interactivespaces.mud.actions.ChangeModulePropertiesAction">
          <properties>
            <property key="moduleId">bt_module</property>
            <property key="search">true</property>
          </properties>
        </action>
      </handler>
      <handler event="net.interactivespaces.mud.events.ChoiceSelectedHandler">
        <guard>
          <properties>
            <property key="choice">mainmenu.choice2</property>
          </properties>
        </guard>
        <action name="net.interactivespaces.mud.actions.EndDramaAction">
                  </action>
      </handler>
      <handler event="net.interactivespaces.mud.events.ChoiceSelectedHandler">
        <guard>
          <properties>
            <property key="choice">mainmenu.choice1</property>
          </properties>
        </guard>
        <action name="net.interactivespaces.mud.actions.SetInModelAction">
          <properties>
            <property key="key">bt_searching</property>
            <property key="value">yes</property>
```

```
            </properties>
          </action>
          <action name="net.interactivespaces.mud.actions.ShowInformationAction">
            <properties>
              <property key="message">mus.guide.msg</property>
              <property key="title">mus.guide.title</property>
              <property key="imageHandle">guide</property>
            </properties>
          </action>
          <action name="net.interactivespaces.mud.actions.ChangeModulePropertiesAction">
            <properties>
              <property key="moduleId">bt_module</property>
              <property key="search">true</property>
            </properties>
          </action>
        </handler>
      </eventhandlers>
    </node>
  </nodes>

</drama>
```

Eksempel 35: Museumshistoriens XML

9.3 Arduino-implementering

De eksterne enheder er beskrevet i afsnit 5.1.1 *Design af historien* og koden til de forskellige enheder (i prototypen) kan findes på den vedlagte CD i mappen `/Arduino` eller på nettet på http://speciale.barmonger.org/arduino.php.

Implementeringen af Arduino-enhederne er beskrevet herunder.

9.3.1 Softwareimplementering

Softwaren på Arduino-enheder er implementeret som beskrevet i afsnit 4.2 (*Konfigurationsprotokol på Arduino*) og 4.3.3 (*Implementering på Arduino-platformen*) og alle de eksterne enheder benytter sig af det framework, der er beskrevet i afsnit 4.4 (*Implementering af Arduino-framework*).

Softwaren adskiller sig ikke fra frameworket, bortset fra Rumsensor-enheden, der aflæser analoge værdier fra Arduino-boardet.

Analoge aflæsninger

For at de analoge enheder ikke står og trækker strøm konstant, er de koblet til en digital udgang, som benyttes til at tænde og slukke for spændingen (se afsnit 9.3.2 *Hardwareimplementering* nedenfor). Hver gang en applikation beder om en analog aflæsning sættes det digitale ben højt, målingen foretages og benet sættes lavt igen.

```
digitalWrite(PIN, HIGH);
delay(SHORT_WAIT);
//Aflæs analog værdi
digitalWrite(PIN, LOW);
```

Lyssensoren har måleværdier der ligger i intervallet 0-1023, hvor 0 er intet lys og 1023 er maksimal lyspåvirkning. XML-beskrivelsen angiver at der skal være lysværdier fra 0 og op til og med 100. Derfor divideres den aflæste analoge værdi med 10.24, for at få værdien til at passe til det rigtige interval. Det har ikke været muligt at lave objektive målinger på lysstyrken, for at finde frem til en passende omregningsfunktion, da lysstyrken er et forholdstal mellem et uspecificeret maksimum og minimum. Derfor benyttes kun denne simple omregning.

```
int light = (int)((float)analogRead(LIGHT_IN) / 10.24);
```

Temperatursensoren er mere kompliceret, da det er nødvendigt at lave målinger for forskellige temperaturer for at finde frem til en funktion, der kan omregne fra måleværdi til temperatur. Temperatursensoren har, på samme måde som lyssensoren, et værdiinterval på 0-1023 som skal mappes til de rigtige temperaturværdier.

Følgende målinger er blevet foretaget for at kalibrere sensoren.

Temperatur (°C)	Måling
-10	250
10	380
21	450
37	600

Tabel 10: Temperaturmålinger

Herefter blev en funktion til at udregne temperaturen (når man kender måleværdien) fundet vha. regression. Regressionen blev udført ved at benytte Wolfram Alpha-tjenesten[15]. Her kunne det ses at den kvadratiske regression var det bedste fit til de målte værdier.

Den resulterende omregningsfunktion for temperaturen er (Y er temperaturen og X er den aflæste analoge værdi): $Y = -0.00011549\ x^2 + 0.233247\ x - 61.2495$.

Figur 53 herunder viser et plot af funktionen.

Figur 53: Kvadratisk regression af temperaturomregningsfunktionen

Funktionen implementeres på Arduino på denne måde:

```
float x = (float)analogRead(TEMP_IN);
float y = (-0.00011549 * x * x) + (0.233247 * x) - 61.2495;
return (int)y;
```

9.3.2 Hardwareimplementering

Som beskrevet i afsnit 4.1.1 (*Hardware*) benyttes der ArduinoBT-boards[16] til denne prototype. De stiller en lang række digitale ind-/udgange og analoge indgange til rådighed for tilføjelse af ekstern hardware. I denne prototype er der 2 enheder, der benytter sig af ekstra hardware; MiniKvarium og Rumsensor.

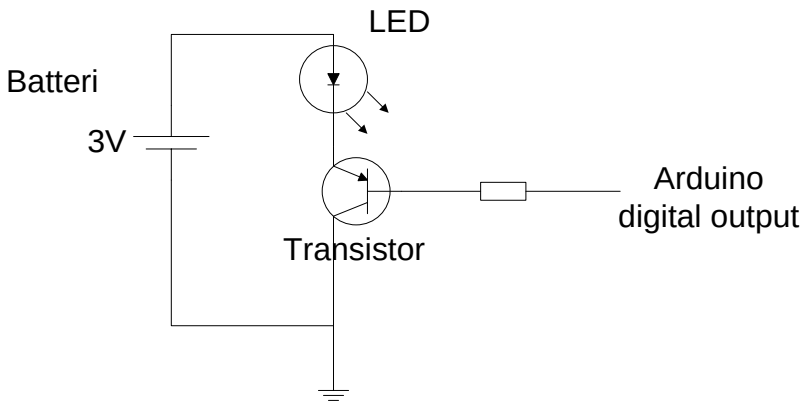
MiniKvarium

Denne enhed er forbundet til et batteridrevet kunstigt miniakvarium. Akvariet har to elementer der kan styres udefra, en LED, der lyser ind i akvariet og en motor der driver vandet rundt og skaber bevægelse deri. Begge dele er forbundet til en kontakt.

I denne prototype er der blevet tilføjet en transistor, som bliver styret af en ekstern spænding. Når transistoren er åben, sendes der strøm udenom den fysiske kontakt og den tilsluttede enhed starter. I Figur 54 er vist et eksempel på dette kredsløb, hvor det er akvariets LED der er tilsluttet. Samme kredsløb er implementeret for elmotoren. Når der er spænding (5V) på Arduino-enhedens digitale output, åbner transistoren og kredsløbet er sluttet, så LED'en kan lyse.

[15] http://www.wolframalpha.com/input/?i=fit+{250%2C+-10}%2C{380%2C+10}%2C{450%2C+21}%2C{600%2C+37}
[16] http://arduino.cc/en/Main/ArduinoBoardBluetooth

De digitale udgange på Arduino-enheden er dem, der er beskrevet som aktuatorer i XML-beskrivelsen. Derved er det muligt for en bruger af systemet at tænde og slukke for lys og bevægelse i akvariet, fra en mobiltelefon.



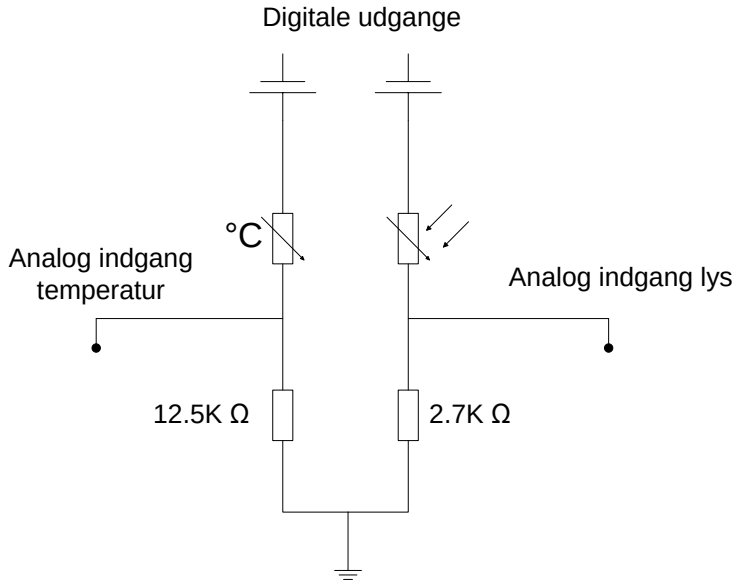
Figur 54: Kredsløb til styring af MiniKvariets lys

Rumsensor

De to sensorer på rumsensor-enheden består af en temperaturfølsom og en lysfølsom modstand. Disse to komponenter har en elektrisk modstand der falder når henholdsvis temperaturen eller lysniveauet stiger. Dette udnyttes i et simpelt kredsløb, som kan ses i Figur 55.

De to faste modstande er valgt ud fra det ønskede område, hvor de to sensorer skal være mest følsomme. Kredsløbet fungerer ved at spændingen over de variable modstande falder, når modstanden falder. Derved stiger spændingen over de faste modstande og det er den spænding vi måler på de analoge indgange.

Når spænding på den analoge indgang stiger, ved vi at det er fordi temperaturen/lysniveauet er steget tilsvarende. Spændingen over hele systemet styres ved at tænde og slukke for de digitale udgange.



Figur 55: Temperatur- og lysmåler

9.4 Verifikation og test af mobilapplikationen

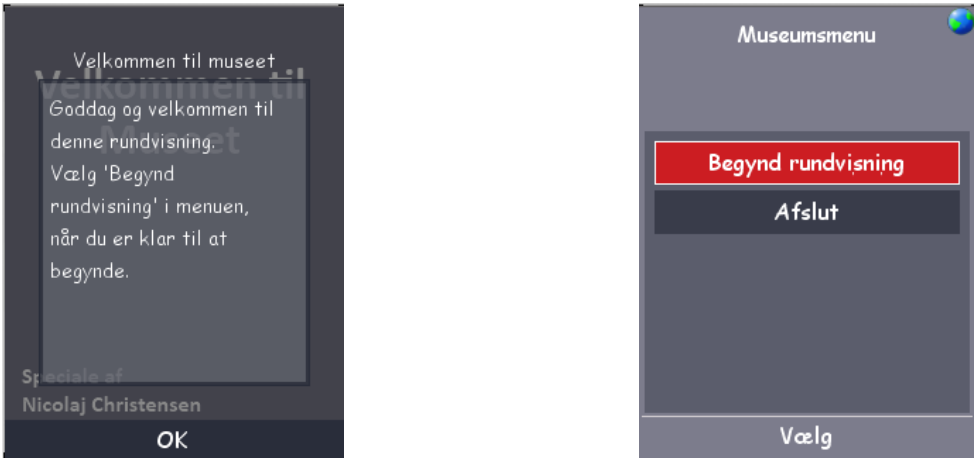
En beskrivelse af implementeringen og en diskussion af resultaterne kan ses i afsnit 5.2.2.

Alle skærmbilleder af applikationen (i denne verifikation) kommer fra den emulator, der blev benyttet under udviklingen[17], medmindre andet er angivet.

Programmet

Første trin i verifikationen er at undersøge hovedprogrammets opførsel ved kun at se på den generelle opførsel, der er uafhængig af de specifikke eksterne enheder.

Velkomstskærmen og hovedmenuen i programmet giver brugeren en simpel og overskuelig indgang til programmet. De to skærme formidler de forventede informationer.



Figur 56: Velkomst og hovedmenu

Når brugeren starter en søgning vises der en simpel guide, der samtidig fungerer som "venteskærm" når programmet ikke foretager sig noget.

Figur 57: Guide til søgning

[17] Sony Ericsson SDK 2.5.0.6 for the Java™ ME Platform
http://developer.sonyericsson.com/wportal/devworld/technology/java/sdk

Guiden viser brugeren hvordan programmet benyttes, ved at billedet viser hvordan (det er en animation hvor telefonen bevæger sig hen til maleriet) og teksten giver en kort vejledning. Guiden bliver vist automatisk hver gang en rundvisning startes, som forventet.

Visningen af fundne enheder kan ses i Figur 58 (venstre). Hvis brugeren svarer nej til at se udstillingen vises en oversigt over udstillinger indenfor rækkevidde af telefonen (Figur 58 (højre)). Begge dialoger giver brugeren relevante muligheder for at komme videre i programmet/rundvisningen.

Figur 58: Eksterne enheder er fundet

Download af ressourcer er implementeret så dette fungerer i baggrunden efter at brugeren har startet en udstilling. Mens dette foregår vises brugeren det skærmbillede der kan ses i Figur 59.

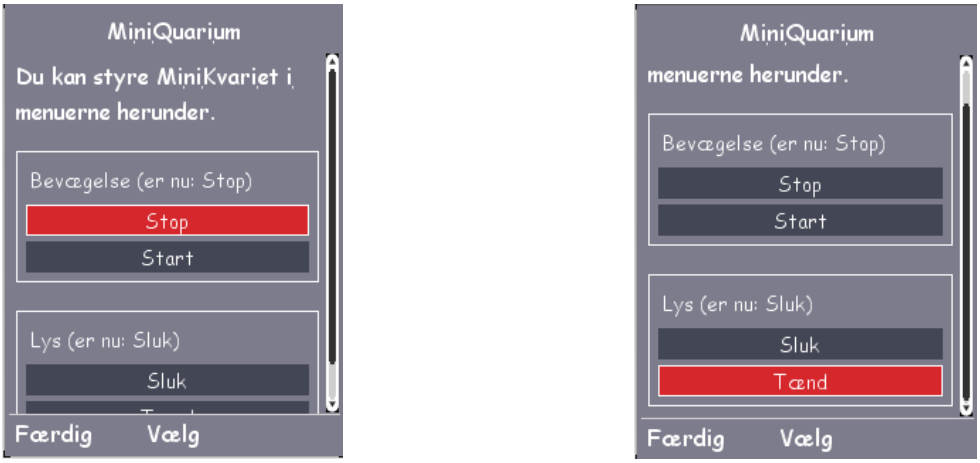
Figur 59: Eksempel på download af ressourcer til en udstilling

MiniKvarium

Introskærmen til MiniKvariet er en simpel billedskærm med tekst, der vises til brugeren når udstillingen startes. Der er mulighed for at brugeren kan navigere til den interaktive del af udstillingen.

Figur 60: MiniKvarium introskærm

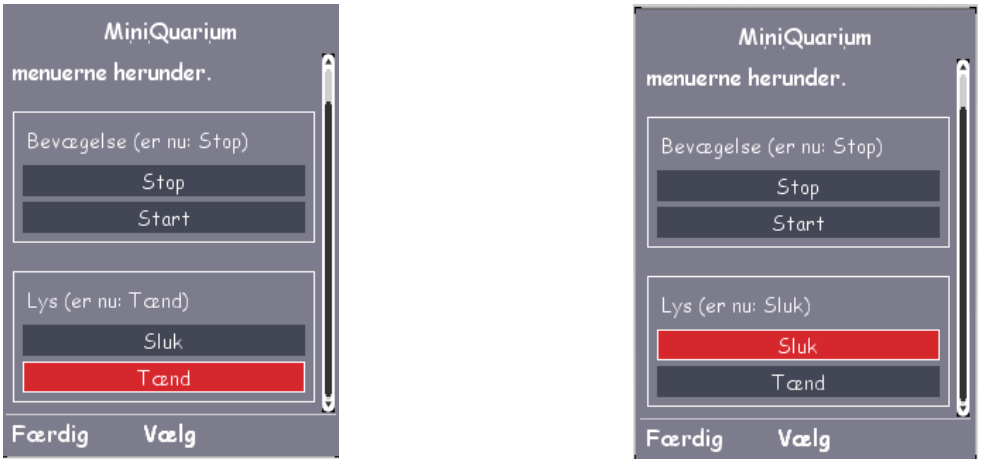
Herefter vises brugeren en oversigt over de interaktive muligheder som MiniKvariet stiller til rådighed. Menuen er vist i Figur 61, hvor det venstre billede viser den øverste del og det højre billede viser skærmbilledet når brugeren har scrollet ned i bunden.



Figur 61: MiniKvarium-menuen

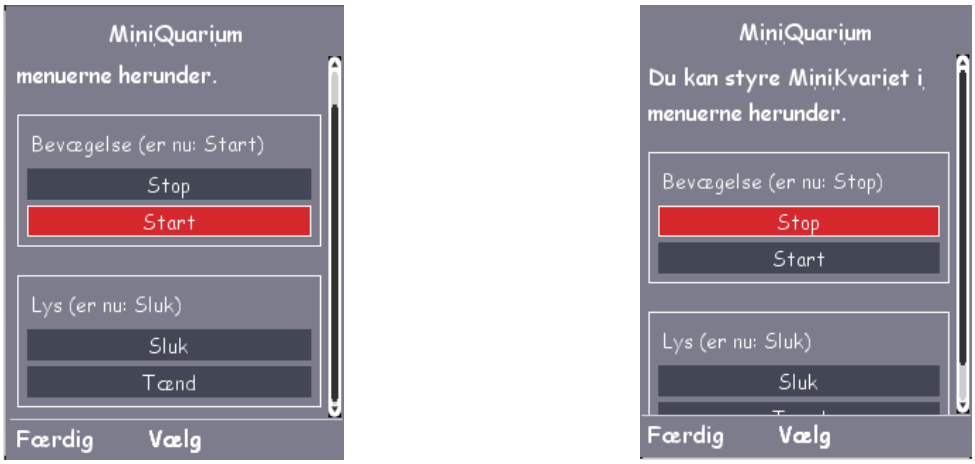
Som forventet har applikationen indlæst XML-filen korrekt og viser brugeren de samme muligheder som der er beskrevet i XML-beskrivelsen (En 'Bevægelse'-aktuator med to tilstande [Start og Stop] og en 'Lys'-aktuator ligeledes med to tilstande [Sluk og Tænd]).

Herefter er det muligt at aktivere en knap og ændre værdien af en af de to tilstandsaktuatorer. Værdien i 'er nu:'-feltet opdateres automatisk til den værdi der er aktiv på den eksterne enhed.



Figur 62: Tænd og sluk for lys i MiniKvariet

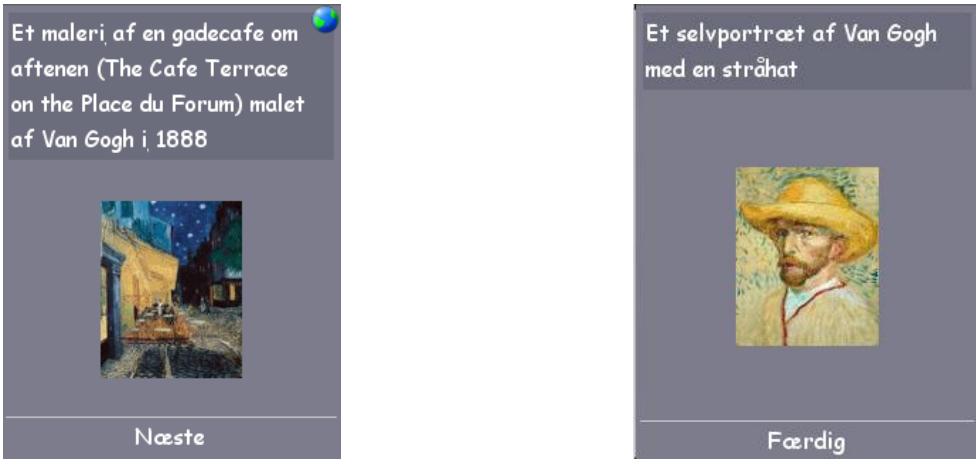
Det samme gør sig gældende hvis brugeren ønsker at ændre tilstanden af 'Bevægelse' i MiniKvariet. Knapperne ændrer tilstanden på den eksterne enhed og tekstfeltet opdateres automatisk til at vise enhedens nye tilstand.



Figur 63: Start og stop for bevægelse

Van Gogh

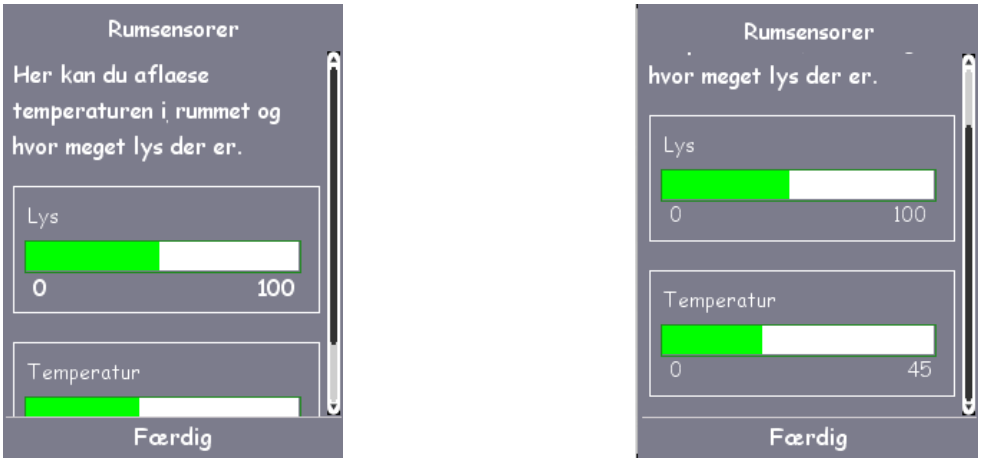
Skærmbillederne i Figur 64 viser at historieudvidelsen er blevet indlæst korrekt fra den eksterne enhed.



Figur 64: Van Gogh-udstillingens første og sidste billede

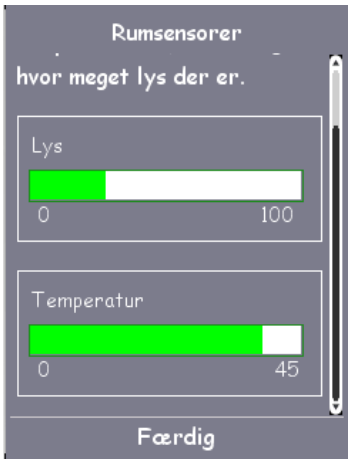
Rumsensor

Som det kan ses i Figur 65 er XML-beskrivelsen blevet indlæst korrekt og begge sensorer er vist på skærmen, med værdier aflæst fra den eksterne enhed. Temperatur- og lyssensorerne viser også de korrekte minimum- og maksimumværdier.



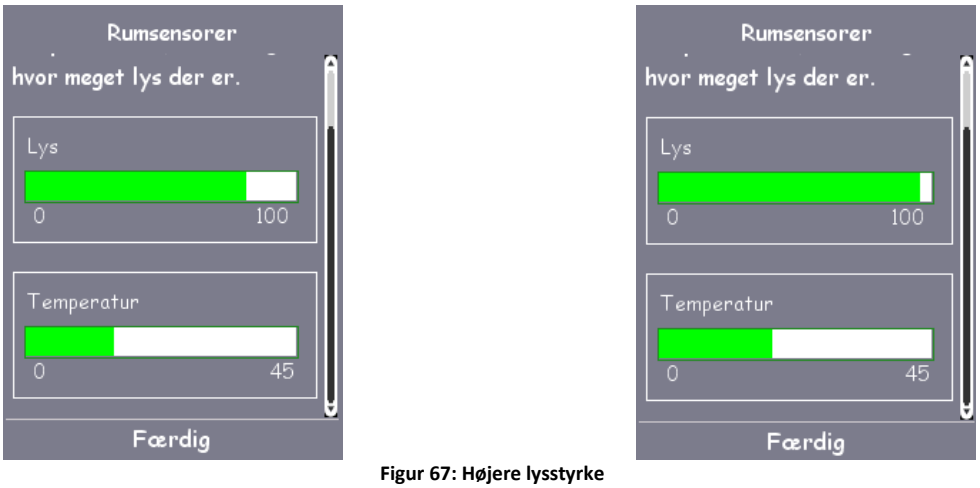
Figur 65: Rumsensorudstillingen

Den eneste mulighed, der gives brugeren, er at afslutte visningen af udstillingen, da opdatering af de to sensorer håndteres automatisk. Herefter øges temperaturen for enheden og værdien i brugerfladen opdateres automatisk (se Figur 66).



Figur 66: Temperaturen stiger

Endeligt øges mængden af lys hvilket bliver indikeret på lyssensoren i mobilapplikationen, der automatisk opdateres hver femte sekund (Figur 67).



Figur 67: Højere lysstyrke