Dokumentácia k inžinierskemu dielu

FIIT KINECT

Tím č. 11

Členovia tímu:

Bc. Ivana Bohunická
Bc. Ján Greppel
Bc. Juraj Muránsky
Bc. František Nagy
Bc. Dominik Rerko
Bc. Matúš Ujhelyi
Bc. Zuzana Ujhelyiová

Tímový projekt 2012/2013
Fakulta Informatiky a Informačných technológií
Slovenská Technická Univerzita

Obsah:

1 Úvod

1.1 Účel dokumentu

Dokument obsahuje dokumentáciu k systému, ktorý je určený na ovládanie rôznych koncových zariadení pomocou senzoru Kinect podľa zadania projektu. Systém je vyvíjaný v rámci predmetu Tímový projekt ako časť inžinierskeho štúdia na Fakulte informatiky a informačných technológií STU v Bratislave. Zadávateľom projektu je Ing. Michal Kottman.

1.2 Štruktúra dokumentu

Vývoj systému je riadený metodikou *Scrum*, a tomu je prispôsobená aj štruktúra tohto dokumentu. Každá kapitola reprezentuje jeden šprint. Podkapitoly reprezentujú úlohy, ktoré členovia tímu v danom šprinte vykonávali. Každá úloha sa, podľa potreby, skladá zo samostatnej analýzy, špecifikácie, návrhu, implementácie a testovania.

1.3 Zadanie projektu

Senzor Kinect dáva technickú možnosť získavania multimodálnej informácie z rôznych zdrojov, obrazových i zvukových. Pre túto úlohu sú dôležité predovšetkým dáta obsahujúce 3D obrazovú informáciu (hĺbkovú mapu) ako vstup pre modul určený na detekciu a rozpoznávanie pohybu používateľa (gest) v reálnom čase. Výstupom projektu má byť softvérová aplikácia, ktorá umožní budúcemu používateľovi – HCI experimentátorovi, definovať vlastné gestá, či udalosti a priradiť im komunikačné povely pre rôzne zariadenia (napr. prostredníctvom IrDA, Bluetooth, WiFi). Aplikácia ďalej umožní vytvárať zostavy komunikačných povelov riadených gestami a poskytne tiež možnosť navrhnutý experimentálny súbor testovať v reálnych podmienkach.

Študenti sa oboznámia s technológiami z počítačového videnia a počítačovej grafiky v praktickej aplikácii, vytvoria modulárny softvérový nástroj s použitím existujúcich softvérových modulov z predchádzajúceho tímového projektu.

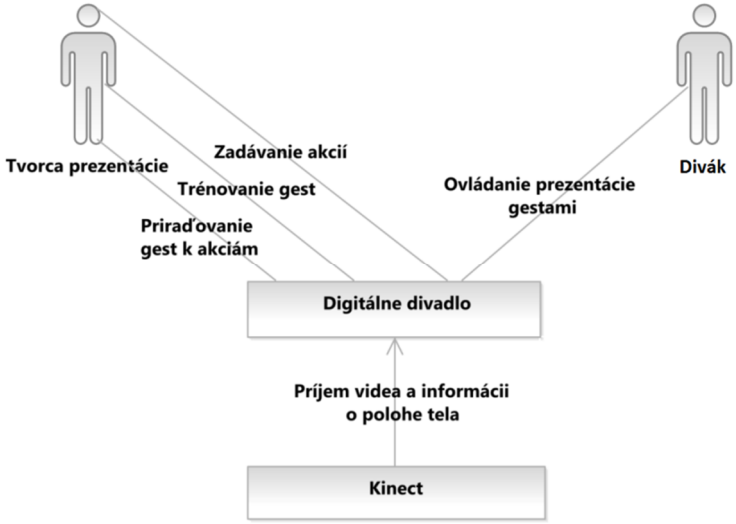
Do projektu je zahrnutý i vývoj algoritmov na rozpoznávanie objektov a gest ako aj komunikácia s externými zariadeniami. Úlohou tímu bude vytvoriť konfigurovateľný softvérový experimentálny nástroj pre návrh a testovanie metód prirodzeného ovládania domácich elektroprístrojov (ako napr. TV prijímač, HiFi súprava a iné) prostredníctvom gest.

2 Úlohy vykonávané pre začatím šprintov

2.1 Analýza riešenia tímového projektu Digitálne divadlo (2011)

Cieľ a hlavná funkcionalita

Cieľom práce bolo analyzovať možnosti dostupných prostriedkov na prácu s obohatenou realitou s použitím ovládača Kinect od spoločnosti Microsoft a navrhnúť funkčné riešenie takéhoto systému. Systém umožňuje tvorbu interaktívnej prezentácie a ovládanie pomocou gest snímaných kamerami ovládača Kinect. K systému je spísaná aj používateľská príručka. Obrázok 2-1 popisuje Tvorcu prezentácie a Diváka ako dvoch základných aktérov v systéme.



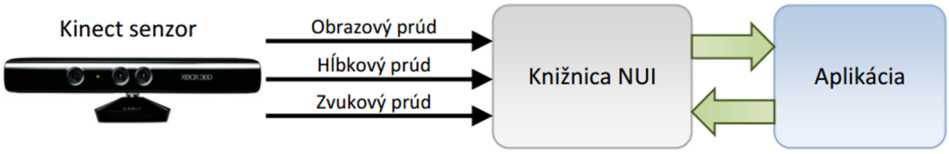
Obrázok 2-1:Kontext systému

Používateľ aplikácie si môže sám zvoliť aké gestá chce v režime prezentácie používať. Aplikácia poskytuje možnosť trénovania nových gest, kedy používateľ bude rovnaké gesto opakovať dostatočný počet krát. Naučený pohyb sa zapamätá a bude ním možné spúšťať istú funkciu pri prezentovaní. V záujme jednoduchého použitia a zvyšovania spoľahlivosti aplikácie, je možné jednotlivé gestá upravovať ich dotrénovaním. Dôležitou funkciou je aj budovanie databázy doteraz natrénovaných gest, aby používateľ nemusel pri každom novom projekte trénovať všetky gestá odznovu.

Akciami sa myslia rôzne udalosti, ktoré môžu byť vyvolané gestami. Aplikácia podporuje videozáznamy, samostatné audio nahrávky a obrázky. Jednotlivé multimediálne súbory sa dajú pridať do projektu a následne v režime tvorby prezentácie priraďujú ku gestám.

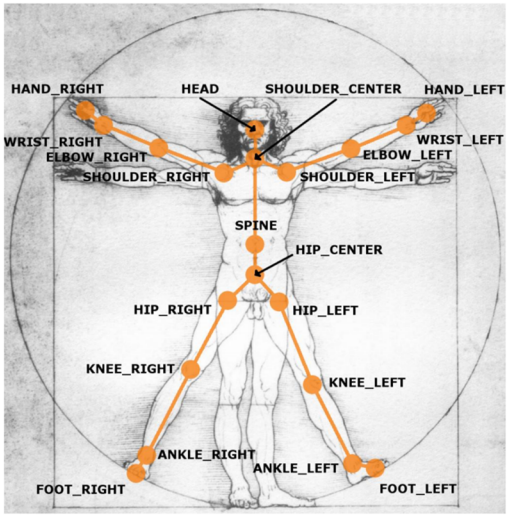
Kinect SDK

Kinect SDK beta od Microsoft Research je nástroj určený pre vývoj aplikácií spolupracujúcimi so senzorom Kinect. Obsahuje ovládače a poskytuje rozhranie pre tvorbu aplikácií a rozhranie zariadenia. Interakcia medzi hardvérom a softvérom s aplikáciou je znázornená na Obrázok 2-2.



Obrázok 2-2:Interakcia hardvéru a softvéru s aplikáciou

NUI API, ktoré je súčasťou Kinect SDK, poskytuje informácie o polohe jedného alebo dvoch hráčov stojacich pred senzorom Kinect spolu s detailnými informáciami o ich pozícii a orientácii. Údaje sú dodávané aplikácii vo forme množiny bodov (nazývaných body kostry), ktoré vytvárajú kostru (Obrázok 2-3).



Obrázok 2-3: Pozície kĺbov kostry relatívne k ľudskému telu

Návrh do modulov, ich zodpovednosti a hierarchia tried

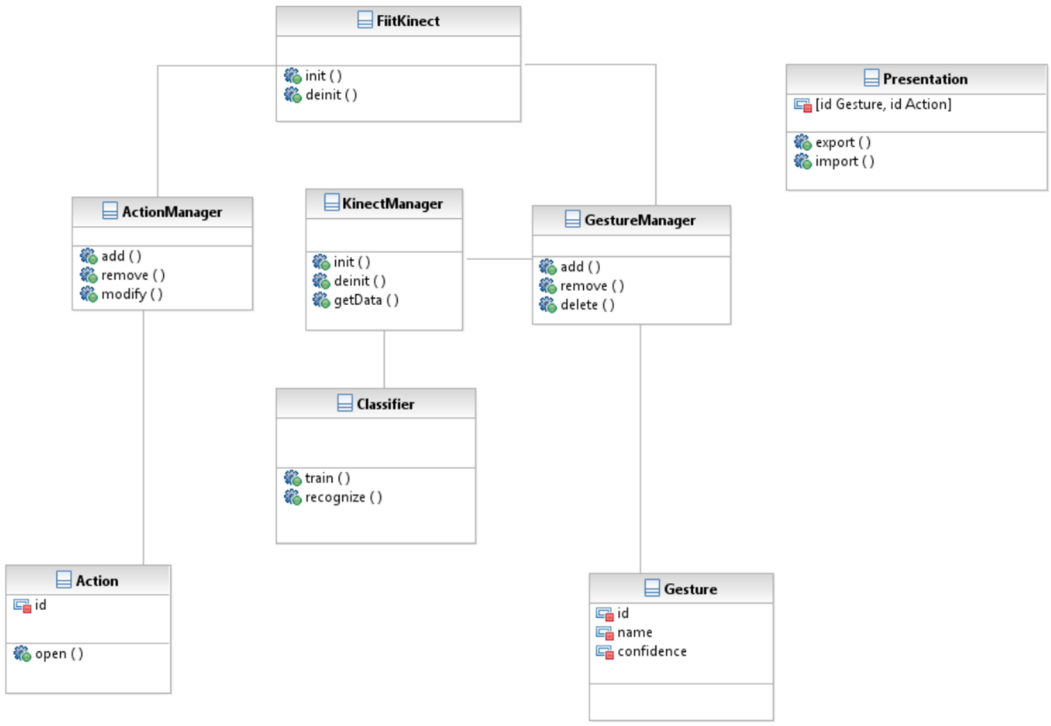
Ako popisuje Obrázok 2-4, aplikácia sa skladá z troch hlavných modulov. Jadrom programu je knižnica FiitKinect, ktorá vykonáva všetky dôležité funkcie a komunikuje s ostatnými modulmi. Na najnižšej úrovni sa nachádza zariadenie Kinect, ktoré nám dodáva potrebné informácie o nasnímanom objekte. S týmto zariadením komunikuje iba vyvjaná knižnica. Používateľské rozhranie je na najvyššej úrovni a od knižnice FiitKinect je úplne oddelené. Jeho funkcia je čisto pre zobrazovacie účely.

Obrázok 2-4: Blokový návrh riešenia

Softvérový návrh aplikácie pozostáva z 8 modulov, z ktorých každý má jasne definovanú zodpovednosť.

Trieda	**Zodpovednosť**
FiitKinect	tvorí rozhranie medzi grafickým rozhraním aplikácie a jej vnútorným spracovaním
KinectManager	nachádza sa na najnižšej úrovni, komunikuje priamo s hardvérovým zariadením Kinect; slúži ako rozhranie medzi Kinectom a tvorenou aplikáciou
Classifier	rozpozná alebo natrénuje gesto
Gesture	uchováva trénovacie dáta toho-ktorého gesta a poskytuje základné operácie s gestom (export a import)
GestureManager	umožňuje pridanie, zmazanie a dotrénovanie gest, ako aj hromadný export/import
Action	uchováva dáta o akciách; akcia je dvojica (priradenie) gesto:akcia
ActionManager	spravuje akcie
Presentation	reprezentuje samotnú prezentáciu

Obrázok 2-5 znázorňuje hierarchiu týchto tried.



Obrázok 2-5: Hierarchia tried

Poznámka ku kapitole: diagramy a texty boli vybrané z projektovej dokumentácie tímového projektu Digitálne divadlo.

2.2 Základná koncepcia Connection Kin

Základnú koncepciu riešenia vidíme v modulárnom systéme skladajúcom sa z hlavného modulu, ktorý by bežal na osobnom počítači. Tento hlavný modul by bol schopný:

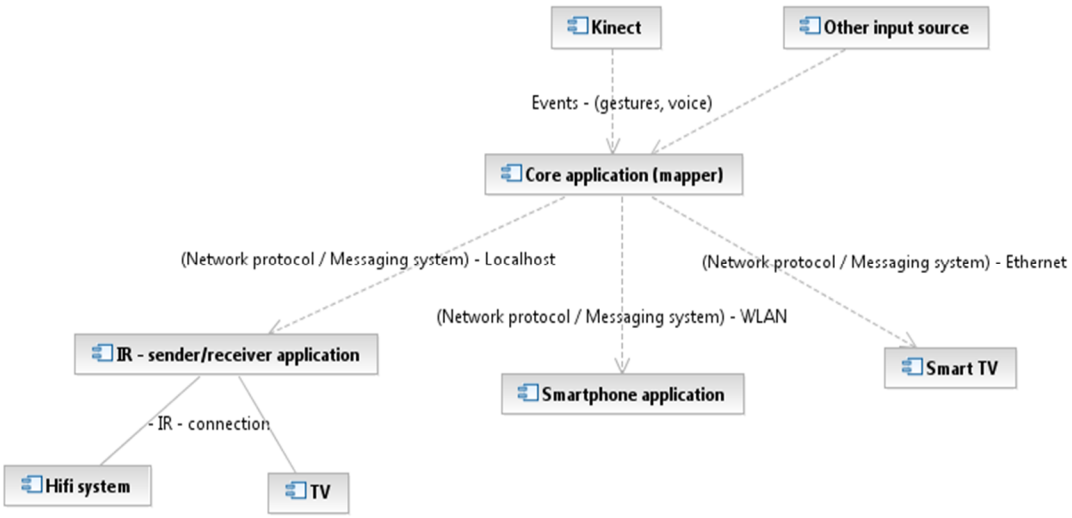
- prijímať podnety vo forme gest alebo zvukových povelov zo senzora Kinect,
- notifikovať iné moduly (klientské moduly) a komunikovať s nimi cez štandardné sieťové protokoly na základe definovaných správ.

Takéto rozdelenie poskytne možnosti pre budúce rozšírenie inými modulmi, ktoré by boli vlastne len novými aplikáciami, pripájajúcimi sa na hlavnú aplikáciu. Poskytne sa tak možnosť zdieľania používateľského prostredia, akým by bol Kinect, pre rôzne platformy a zariadenia. Jednotné gestá alebo zvukové povely použité pre viaceré zariadenia sú spôsobom ako ovládať rôzne druhy zariadení. Zabezpečenie jednotnosti prenáša znalosť „ako ovládať zariadenie“ z používateľa na tvorcov systému.

Každý klientský modul:

- zabezpečuje proces ovládania jednotlivých zariadení,
- je oddelený od hlavného modulu, čo vylepšuje stabilitu i reakcie systému v reálnom čase,
- sa môže sám prispôsobiť zariadeniu, na ktorom bude bežať alebo s ktorým bude komunikovať.

Zamýšľanú koncepciu vyjadruje Obrázok 2-6.



Obrázok 2-6: Koncepcia riešenia - FIIT Kinect

Jedným z možných riešení mapovania jednotlivých gest je napevno nadefinovať gestá a vyberať si zariadenia na ovládanie ako aj akciu vykonateľnú na zariadení. Inou možnosťou je navrhovanie vlastných gest používateľom a následným mapovaním týchto gest na jednotlivé akcie. Definovanie vlastných udalostí, ktoré je možné vykonávať na zariadeniach ale považujeme sa špecificky závislé na klientských moduloch.

2.3 Product Backlog

2.3.1 Zoznam používateľských príbehov

Vývoj systému prebieha agilnou metódou Scrum, kde je ako prvá fáza definovanie požiadaviek vo forme používateľských príbehov. Príbehy sú pre lepšie pochopenie a komunikáciu písanie jazykom používateľa. Pri niektorých príbehoch boli identifikované aj jednoduché prípady použitia. Nasleduje zoznam používateľských príbehov rozdelených podľa tematík.

Komunikácia modulov po sieti

Rola: používateľ

Funkcionalita: jednotlivé moduly (zdroje udalostí a cieľové zariadenia) komunikujú po

sieti

Benefit:

1. mám možnosť ovládať zariadenia vzdialene (po sieti)
2. systém je stabilnejší (chyby v moduloch nespôsobia pád celého systému)

Prípad použitia:

Zahájenie spojenia

1. spustím centrálny modul na PC1
2. spustím modul pre TV a HiFi na PC2
3. spustím modul pre Kinect na PC3
4. centrálny modul hlási pripojenie obidvoch modulov

Komunikácia medzi modulmi

1. modul pre Kinect rozpozná gesto a pošle ho centrálnemu modulu
2. centrálny modul
   a) prijme správu a
   b) komunikuje modulu pre TV a Hifi nejakú akciu (napr. zvýšenie hlasitosti)
3. modul pre TV a HiFi vykoná akciu (zvýši hlasitosť)

2.3.2 Gestá

2.3.2.1 Pridávanie a rozpoznávanie gest

Rola: používateľ

Funkcionalita: viem pridať nové gesto, ktoré Kinect vie následne rozpoznať

Benefit: mám možnosť následne mapovať toto naučené gesto na nejakú akciu

Prípad použitia:

1. zvolím si možnosť pridať gesto
2. systém mi dá
   a) 5 sekúnd na prípravu a
   b) 3 sekundy na spravenie nového gesta
3. spravím gesto

4. systém

   a) zaznamená gesto

   b) vypíše správu o úspechu

   c) ponúkne možnosť pre zadanie mena pre naučené gesto

5. zadám meno pre gesto, potvrdím zadanie

6. ak spravím gesto ešte raz

   a) rozpozná ho a

   b) vypíše meno gesta, ktoré rozpoznal

2.3.2.2 Módy zariadení

Rola: konfigurátor, používateľ

Funkcionalita: určitým gestom sa môžem prepnúť do "módu zariadenia" (napr. módu TV, módu HiFi)

Benefit: tie isté gestá môžem využiť pre rôzne typy zariadení (napr. zvýšiť hlasitosť môže tým istým gestom pri TV ako aj pri HiFi)

Prípad použitia:

Nastavenie módu (konfigurátor)

1. gesto-1 nastavím ako TV mód
2. gesto-2 nastavím ako HiFi mód
3. gesto-3 nastavím pre zvýšenie hlasitosti

Prepnutie módu (používateľ)

1. spravím gesto-1, potom gesto-3
2. systém zvýši mi hlasitosť na TV
3. spravím gesto-2, potom gesto-3
4. systém zvýši mi hlasitosť na HiFi

2.3.2.3 Spätná väzba

Rola: používateľ

Funkcionalita: vidím a) či bolo gesto korektne rozpoznané

   b) v akom móde sa práve nachádza

c) aké akcie boli vykonané

Benefit: mám spätnú väzbu o dianí v systéme

2.3.3 Mapovanie

2.3.3.1 Vytváranie mapovania

Rola: konfigurátor

Funkcionalita: mám možnosť mapovať udalosti na akcie

Benefit: môžem rôznymi gestami ovládať externé zariadenia

Prípad použitia:
1. zvolím si udalosť (napr. gesto-5) zo zoznamu
2. zvolím si akciu (napr. zvýšenie hlasitosti na TV) zo zoznamu
3. potvrdím ich namapovanie
4. ak spravím gesto-5, zvýši sa mi hlasitosť na TV

2.3.3.2 Export a import gest, akcií a ich mapovaní

Rola: konfigurátor

Funkcionalita: mám možnosť exportovať gestá, akcie a ich mapovania a znova ich importovať inde

Benefit:
1. pri novej inštalácií nemusím konfigurovať odznova
2. keď sa poškodia údaje alebo ich nechtiac zmením, môžem obnoviť funkčnú zálohu

Prípad použitia:
1. exportujem gestá, akcie a ich mapovania
2. náhodou niečo vymažem a nejde to späť
3. importujem zálohu, ktorú som si spravil
4. všetko ide tak ako pred tým ☺

2.3.3.3 História zmien gest, akcií a ich mapovaní

Rola: konfigurátor

Funkcionalita: mám možnosť pohybovať sa v histórií zmien gest, akcií a ich mapovaní

Benefit: v prípade chyby môžem vrátiť naspäť zmenu, resp. vrátiť stav do istého časového momentu

2.3.4 Iné udalosti

2.3.4.1 Príkazy z mobilu ako udalosť

Rola: používateľ

Funkcionalita: môžem ovládať externé zariadenie cez mobilný telefón

Benefit: môžem ovládať napr. televízor cez mobil a vynechať tak ovládač

2.3.4.2 Zvuk ako udalosť

Rola: používateľ

Funkcionalita: mám možnosť pridať ako udalosť rôzne zvuky (napr. písknutie, tlesknutie, lusknutie)

Benefit: pri niektorých akciách je pre mňa pohodlnejší zvukový vstup (keď idem spať, som v posteli a chcem zhasnúť svetlo, nebudem predsa mávať rukami pred sezorom Kinect)

2.3.4.3 Hlas ako udalosť

Rola: používateľ

Funkcionalita: mám možnosť pridať ako udalosť rôzne slová (napr. "zhasni", "stíš")

Benefit: pri niektorých akciách je pre mňa pohodlnejší hlasový vstup

2.3.5 Iné akcie

2.3.5.1 Ovládanie počítača

Rola: konfigurátor

Funkcionalita: mám možnosť pridať akciu pre spúšťanie programov na PC

Benefit: môžem ovládať počítač (napr. spustiť film, prečítať si poštu)

2.3.5.2 Ovládanie TV a HiFi

Rola: konfigurátor

Funkcionalita: mám možnosť pridať akciu pre vysielanie infračervených signálov pre ovládanie TV a HiFi

Benefit: môžem ovládať televízor a HiFi

2.3.5.3 Ovládanie telefónu

Rola: konfigurátor

Funkcionalita: mám možnosť pridať akciu pre ovládanie telefónu

Benefit: môžem ovládať telefón (napr. vypnúť zvuky, odmietnuť hovor)

3 Šprint 1 (Čierna ovca)

3.1 Návrh komunikačného protokolu (CKTP protokol)

Súčasťou zadania témy FIIT Kinect je vytvoriť modulárny nástroj pre ovládanie domácich elektrospotrebičov. Práve z tohto dôvodu bol návrh celého nástroja rozdelený do viacerých komponentov, každý z nich zodpovedný za určitú časť funkcionality systému.

Navyše boli identifikované požiadavky ako možnosť vzdialeného prístupua potreby pripájania a odpájania rôznych elektrospotrebičov za behu systému. Implementácia týchto funkcionalít vyžaduje aby jednotlivé komponenty boli úplne oddelené a komunikovali cez sieťové pripojenie. Aby komunikácia mohla prebehnúť, je potrebné navrhnúť komunikačný protokol.

3.1.1 Typ správ

Správy sú posielané po nadviazaní TCP spojenia medzi klientom a serverom. Úlohou je po nadviazaní spojenia poskytnúť serveru zoznam všetkých akcií (udalostí), ktoré klient poskytuje.

Bolo navrhnutých 7 hlavných typov správ:

- INIT_MESSAGE
- OK_MESSAGE
- PING_MESSAGE
- EVENT_MESSAGE
- ACTION_MESSAGE
- CLOSE_MESSAGE
- INIT_MESSAGE

Obsahom každej správy je

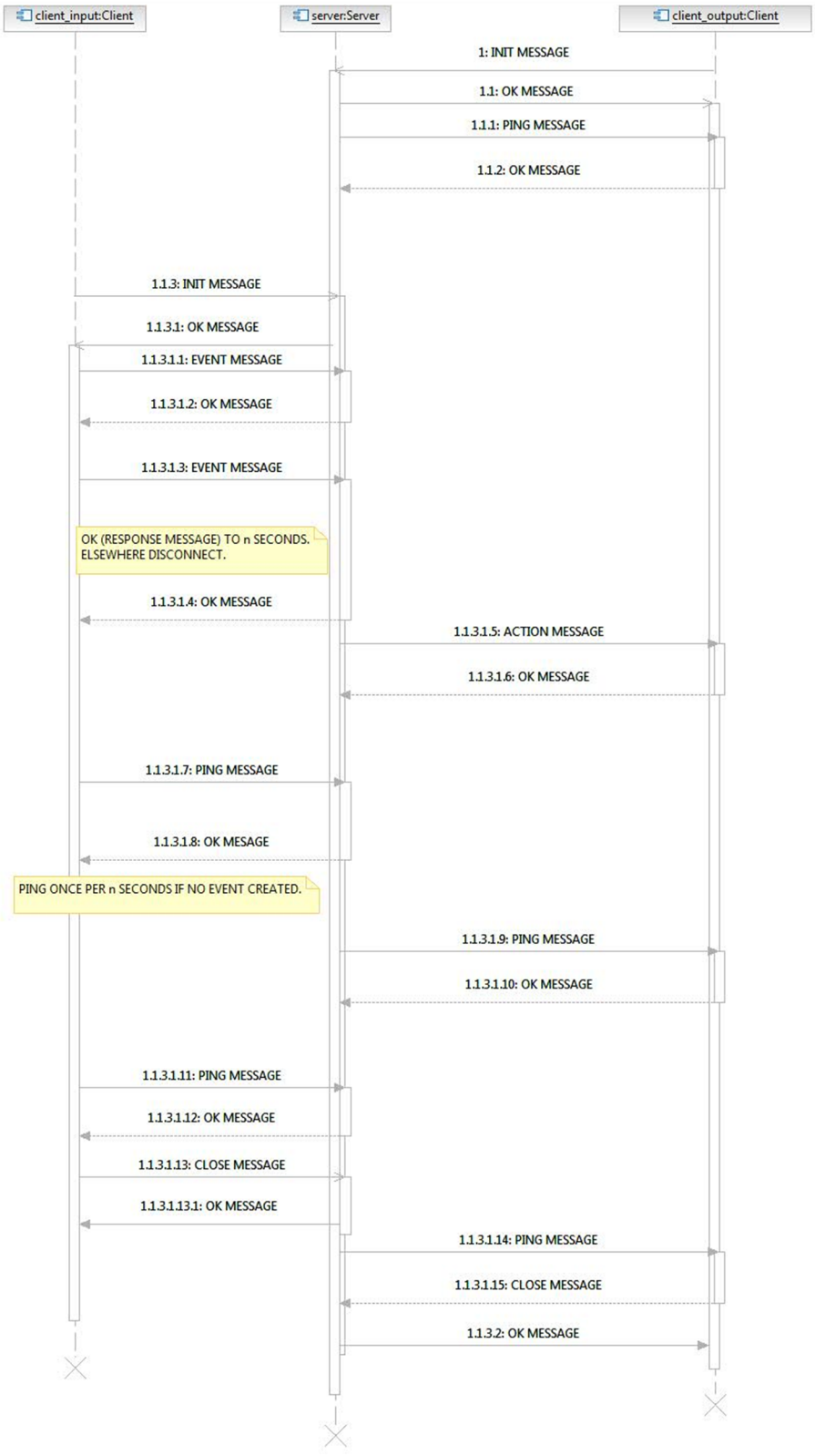
- dĺžka (4B) – predstavuje dĺžku správy, ktorá bude nasledovať,
- typ správy (6B) INIT,
- obsah vo formáte json stringu.

```
{
 "deviceType": "input",
 "idDevice":   "guid",
 "deviceName": "telefon",
 "responseTime":   "5000",
 "pingTime": "15000",
```

```
 "events": [
{
     "eventName":   "gesture27",
     "description": "add volume on device"
  },{
     "eventName":   "gesture37",
     "description": "change channel on TV"
  }
 ]
}
{
 "deviceType": "output",
 "idDevice":   "guid",
 "deviceName": "telka",
 "responseTime":   "5000",
 "pingTime": "15000",
 "actions": [
{
"actionName":"addVolume",
     "description":"add volume on device",
     "value":"2"
  },{
      "actionName":"changeChannel",
      "description":"change channel on TV",
      "value":"2"
   }
]
}
```

V rámci init správy sa pošle buď zoznam akcií (ak ide o prijímač) alebo zoznam udalostí (ak ide o vysielač). Tento zoznam sa vyserializuje do príslušných objektov. Okrem toho sa pošlú parametre spojenia ako response time, čo je čas, do ktorého sa očakáva odpoveď na každú správu a ping time, čo je interval preposielania PING správ, ktoré slúžia len na udržanie spojenia nažive. Tieto časy sú všetky posielané v milisekundách.

3 | Šprint 1 (Čierna ovca)



Obrázok 3-1: Komunikačný diagram protokolu CKTP

3.1.1.1 OK_MESSAGE

Správa sa odosiela ako potvrdenie na prijatie určitých typov správ (PING, CLOSE, INIT, EVENT, ACTION). Neobsahuje žiadnu dátovú časť (žiadny json string ako pri ostatných). Obsah správy je:

- Dĺžka (4B) v tomto prípade vždy 6B
- Typ správy (6B) OK

3.1.1.2 PING_MESSAGE

Správa odosielaná každú minútu od klienta ku serveru v prípade, že nedošlo k žiadnej akcii alebo udalosti. Ide o informáciu, že spojenie je stále živé. Ako odpoveď príde OK_MESSAGE. Obsah správy je definovaný:

- Dĺžka (4B) v tomto prípade vždy 6B
- Typ správy (6B) PING

3.1.1.3 EVENT_MESSAGE

Správa odoslaná od klienta ku serveru v prípade, že nastala udalosť a má sa vykonať príslušná akcia. Obsah správy:

- Dĺžku (4B) dĺžku nasledujúcej správy
- Typ správy (6B) EVENT
- Obsah vo formáte json stringu

Json bude obsahovať meno udalosti, ktorá nastala.

```
{
"eventName":"gesture24"
}
```

3.1.1.4 ACTION_MESSAGE

Predstavuje typ správy odoslaný od servera klientovi v prípade, že nastala nejaká udaľosť. Obsahom je:

- Dĺžka (4B) dĺžka nasledujúcej správy
- Typ správy (6B) ACTION
- Obsah vo formáte json stringu

V rámci json strinu bude uvedený názov akcie, ktorá nastala a value, ktorá môže predstavovať hodnotu o koľko sa má zvýšiť zvuk, alebo ktorý na ktorý kanál sa má televízia prepnúť.

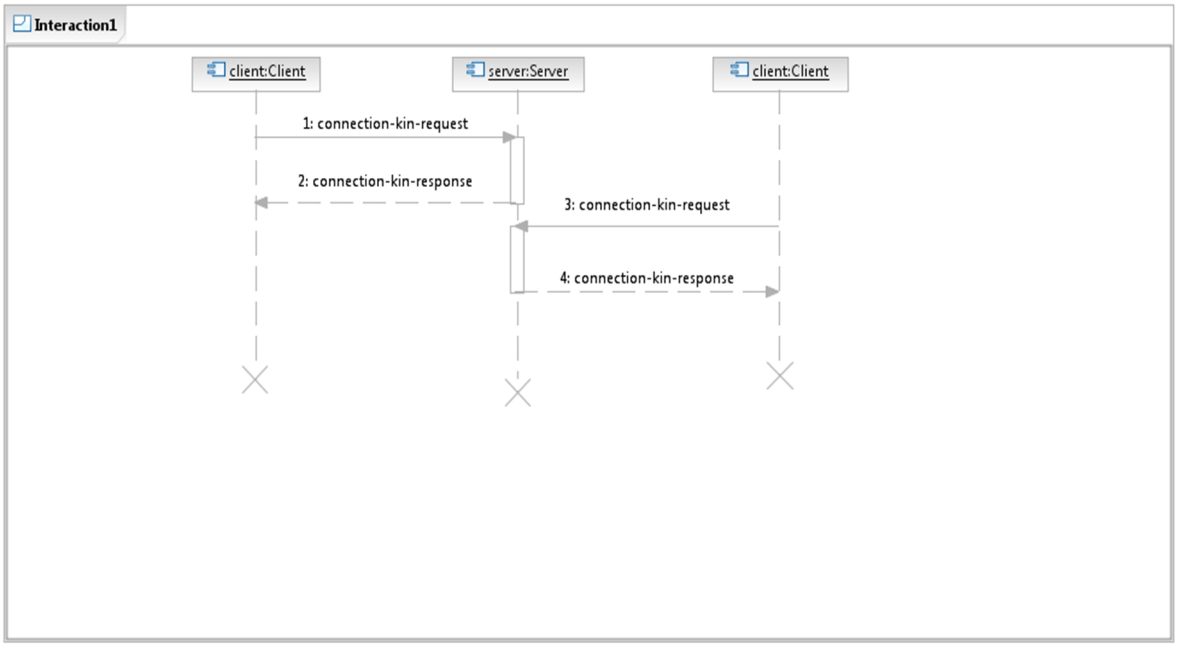
```
{
"actionName":"changeChannel",
"value":"2"
}
```

3.1.1.5 CLOSE_MESSAGE

Správa posielaná od klienta serveru, slúži na ukončenie spojenia. Odpoveďou je OK_MESSAGE.

Formát správy je:

- Dĺžka (4B) dĺžka nasledujúcej správy (v tomto prípade vždy 6)
- Typ správy (6B) CLOSE



Obrázok 3-2: Diagram popisujúci init správu

3.1.2 Zistenie dostupnosti hlavného servera (klient --> server)

3.1.2.1 UDP požiadavka

od	klient
ku	broadcast
transportný protokol	UDP
port vyslania	52726
port doručenia	52726
formát	connection - kin – request

3.1.2.2 UDP odpoveď

od	server
ku	klient
transportný protokol	UDP
port vyslania	52726
port doručenia	52726
formát	connection-kin-response port port_num číslo portu na ktorom server počúva ip_type ipv4/ipv6 ip address-string

3.1.3 Vytvorenie spojenia

3.1.3.1 TCP požiadavka

od	klient
ku	server
transportný protokol	TCP
formát	dĺžka nasledujúcej správy (4B) typ správy (6B) “INIT “ správa vo formáte json dĺžky: dĺžka nasledujúcej správy - 6B (typ správy)

Príklad

4B 6B 260 - 6B

260	INIT _ _	{ "deviceType":"input",
		"idDevice":"guid",
		"deviceName":"telefon",
		"responseTime": "5000",
		"pingTime": "15000",
		"events":[
		{
		"eventName":"gesture27",
		"description":"add volume on device"
		},
		{
		"eventName":"gesture37",
		"description":"change channel on TV"
		}
		]
		}

3.1.3.2 TCP odpoveď

od	server
ku	klient
transportný protokol	TCP
formát	dľžka nasledujúcej správy (4B) typ správy (6B) “OK “

Príklad

4B 6B

6	OK _ _ _ _

3.1.3.3 TCP požiadavka

od	klient
ku	server
transportný protokol	TCP
formát	dĺžka nasledujúcej správy (4B) typ správy (6B) “EVENT “ správa vo formáte json dĺžky: dĺžka nasledujúcej správy - 6B (typ správy)

Príklad

4B 6B 55 - 6B

55	EVENT _	{
		"eventName":"gesture24"
		}

3.1.3.4 TCP odpoveď

od	server
ku	klient
transportný protokol	TCP
formát	dľžka nasledujúcej správy (4B) typ správy (6B) “OK “

Príklad

4B 6B

6	OK _ _ _ _

3.1.3.5 TCP požiadavka

od	klient
ku	server
transportný protokol	TCP

od	klient
formát	dĺžka nasledujúcej správy (4B) typ správy (6B) “PING “

Príklad

4B 6B

6	PING _ _

3.1.3.6 TCP odpoveď

od	server
ku	klient
transportný protokol	TCP
formát	dĺžka nasledujúcej správy (4B) typ správy (6B) “OK “

Príklad

4B 6B

6	OK _ _ _ _

3.1.3.7 TCP požiadavka

od	klient
ku	server
transportný protokol	TCP
formát	dĺžka nasledujúcej správy (4B) typ správy (6B) “CLOSE “

Príklad

4B 6B

6	CLOSE _

3.1.3.8 TCP odpoveď

od	server
ku	klient
transportný protokol	TCP
formát	dĺžka nasledujúcej správy (4B) typ správy (6B) “OK “

Príklad

4B 6B

6	OK _ _ _ _

3.1.4 Zistenie dostupnosti hlavného servera (server <-- klient)

3.1.4.1 UDP požiadavka

od	klient
ku	broadcast
transportný protokol	UDP
port vyslania	52727
port doručenia	52727
formát	connection - kin - request

3.1.4.2 UDP odpoveď

od	hlavný server
ku	klient
transportný protokol	UDP
port vyslania	52727

od	hlavný server
port doručenia	52727
formát	connection-kin-response port port_num číslo portu na ktorom server počúva ip_type ipv4/ipv6 ip address-string

3.1.4.3 TCP požiadavka

od	klient
ku	server
transportný protokol	TCP
formát	dĺžka nasledujúcej správy (4B) typ správy (6B) “INIT “ správa vo formáte json dĺžky: dĺžka nasledujúcej správy - 6B (typ správy)

Príklad

4B 6B 307 - 6B

307	INIT _ _	<pre>{
		"deviceType":"output",
		"idDevice":"guid",
		"deviceName": "telka",
		"responseTime": "5000",
		"pingTime": "15000",
		"actions":[
		{
		"actionName":"addVolume",
		"description":"add volume on device",
		"value":"2"
		},
		{
		"actionName":"changeChannel",
		"description":"change channel on TV",
		"value":"2"
		}
		]
		}</pre>

3.1.4.4 TCP odpoveď

od	server
ku	klient
transportný protokol	TCP
formát	dĺžka nasledujúcej správy (4B) typ správy (6B) “OK “

Príklad

4B 6B

6	OK _ _ _ _

3.1.4.5 TCP požiadavka

od	server
ku	klient

od	server
transportný protokol	TCP
formát	dĺžka nasledujúcej správy (4B) typ správy (6B) “PING “

Príklad

4B 6B

6	PING _ _

3.1.4.6 TCP odpoveď

od	klient
ku	server
transportný protokol	TCP
formát	dĺžka nasledujúcej správy (4B) typ správy (6B) “OK “

Príklad

4B 6B

6	OK _ _ _ _

3.1.4.7 TCP požiadavka

od	server
ku	klient
transportný protokol	TCP
formát	dĺžka nasledujúcej správy (4B) typ správy (6B) “ACTION“ správa vo formáte json dĺžky: dĺžka nasledujúcej správy - 6B (typ správy)

Príklad

4B 6B 71 - 6B

71	ACTION	{
		"actionName":"changeChannel",
		"value":"2"
		}

3.1.4.8 TCP odpoveď

od	klient
ku	server
transportný protokol	TCP
formát	dĺžka nasledujúcej správy (4B) typ správy (6B) “OK “

Príklad

4B 6B

6	OK _ _ _ _

3.1.4.9 TCP požiadavka

od	server
ku	klient
transportný protokol	TCP
formát	dĺžka nasledujúcej správy (4B) typ správy (6B) “PING “

Príklad

4B 6B

6	PING _ _

3.1.4.10 TCP odpoveď

od	klient
ku	server

od	klient
transportný protokol	TCP
formát	dĺžka nasledujúcej správy (4B) typ správy (6B) “CLOSE “

Príklad

4B	6B
6	CLOSE _

3.2 Návrh alternatívneho komunikačného protokolu

V rámci analýzy komunikácie zariadení vznikla potreba navrhnutia komunikačného protokolu. Z dôvodu rôznych pohľadov na daný problém vznikli 2 rôzne alternatívy návrhu protokolu. Hlavná nevýhoda tohto návrhu, ktorá bola aj primárnym dôvodom pre jeho zavrhnutie, je potreba znovu inicializovať spojenie pri každom odosielaní správy od klienta. Z pohľadu obsahu správ posielaných po sieti ako aj reprezentácie dát boli oba návrhy v podstate ekvivalentné. Takisto zoznam udalostí, ktoré poskytuje klient, či akcií, ktoré pozná by v tomto prípade nebol posielaný automaticky - pri pripojení, ale až na vyžiadanie zo strany servera. To by mohlo mať dopad na efektivitu komunikácie.

3.2.1 Komunikácia protokolu

Správy budú posielané cez TCP spojenie pre zaistenie kontroly správneho prenosu dát. Vždy keď je potreba informovať druhú stranu, vytvorí nové spojenie, zašlú sa všetky potrebné informácie a spojenie sa zatvorilí.

3.2.2 Vyhľadanie hlavnej aplikácie

Na zistenie, či v sieti existuje hlavná aplikácia sa použije UDP broadcast, ktorý v sebe nesie informácie o tom, že ide o broadcast kinect aplikácie a na akom porte dané zariadenie počúva odpovede od hlavnej aplikácie. Presný tvar broadcast správy spĺňa následovný formát:

- Kinect_connection [GUID zariadenia]
- IPv[46] [IP addresa]:[port na ktorom sa počúva].

Odpoveď od regulérneho prímača je v tvare:

- „Kinect_connection [GUID tohto prijímača]

- IPv[46] [IP addresa]:[port na ktorom sa počúva]“,

ktorá sa zašle na adresu z broadcastu. Týmto sa jednoznačne identifikuje, kde počúva, ktoré zariadenie.

3.2.3 Zasielanie správ

V pri samotnej komunikácií by sa používali následovné správy:

- CREATE_EVENT,
- CREATE_ACTION,
- ACCEPT,
- REJECT,
- REQUEST_KNOWN,
- KNOWN_EVENTS/ACTIONS,
- SEND_EVENT,
- SEND_ACTION,
- FORGET,
- REQUST_PING,
- PING,
- GOING_OFF.

Každá komunikačná správa pozostáva z json zakódovaných informácií, kde ako prvý je vždy minimálne atribút identifikujúci zariadenie „GUID_device“ a taktiež atribút „receiver_GUID“ identifikujúci príjemcu, aby sa nestalol, že správa bude interpretovaná iným koncovým bodom, ako tým, ktorému bola určená.

Ping správa neobsahuje už žiadne dodatočné informácie. Ostatné komunikačné správy obsahujú názov akcie v atribúte „action“.

Správy typu „create event“ a „create action“

Hlavný objekt má nasledovné parametre:

- „GUID_event“ resp. „GUID_action“
  - Identifikátor, ktorý jednoznačne určuje danú akciu.
- „media“ - nepovinný parameter, má ďalšie parametre, môže sa vyskytnúť viac krát
  - name – názov súboru (nepovinné pole)
  - type – typ definujúci daný súbor

- encoding – spôsob ako sú dáta uložené v súbore (nepovinné pole)
  - momentálne bude podporované len „base64“
- length – nepovinný parameter určujúci dĺžku už zakódovaného súboru (v „encoding“ kódovaní)
- file – samotný prenášaný súbor

Správy typu „accept“

- „GUID_event“ resp. „GUID_action“
  - informácia, že priradenie bolo úspešné a môže sa použiť

Správy typu „reject“

- „message_name“
  - identifikuje, ktorú správu sa nepodarilo zaslať.
- „GUID_event“ resp. „GUID_action“ - ak pôvodná správa tento atribút neobsahuje, tak ho neobsahuje ani reject správa
  - značí, že ktorá akcia alebo udalosť bola odmietnutá.
- „message“ - môže byť prázdny
  - textová správa definujúca v čom nastal problém.
- „message_code“ - nepovinný parameter
  - zaznamenáva kód pre známe chyby. Toto pole bude pozostávať len s identifikačného čísla
    - kód 1 znamená zopakuj správu (pri napríklad neúspešnom transfery súboru),
    - kód 2 znamená obsadený identifikátor pre akciu/udalosť,
    - kód 3 znamená nepochopenie dotazu (napríklad nastal šum v protokole a aplikácia neporozumela správe, alebo prijatá správa nie je definovaná protokolom),
    - kód 4 znamená, že dotaz sa práve nedá vykonať. Toto môže nastať, ak napríklad vysielajúce zariadenie pošle udalosť, pre ktorý momentálne nie je namapovaná žiadna akcia, respektíve na sieti nie je zariadenie, ktoré túto akciu vie obslúžiť.

Správy typu „request known“

- „GUID_event“ alebo „GUID_action” - nepovinný parameter

- určuje spôsob, ako si zariadenie môže vypýtať definíciu akcie alebo udalosti,
- ide o funkčný ekvivalent správy „reject“ s atribútmi „message_name“ a „message_code“ vyplnenými ako „create“ a „1“
- v prípade, že daný atribút nie je zadaný, očakáva sa, že prijímateľ zašle správu typu „known events/actions“

Správy typu „known events/actions“

- „GUID_event“ - atribút sa môže vyskytovať ľubovoľne veľa krát
  - GUID – predstavuje identifikátor známej udalosti,
  - „last_change“ - predstavuje časová značku (unix timestamp) o priradení eventu (kedy u neho táto akcia vznikla)
    - pokiaľ daná akcia už nie je podporovaná, alebo bola predefinovaná na iné správanie, tak nech je o tom server informovaný,
- „GUID_action“ - atribút sa môže vyskytovať ľubovolne veľa krát
  - GUID – predstavuje identifikátor známeho eventu
  - „last_change“ - predstavuje časová značku (unix timestamp) o priradení udalosti (kedy u neho táto akcia vznikla)

Správy typu „send event“ a „send action”

- „GUID_event“ resp. „GUID_action“
  - jednoznačný identifikátor udalosti/akcie, ktorá sa má vykonať,
- „last_change“ - značí časová značku (unix timestamp) o priradení udalosti
  - na základe tejto hodnoty sa prijímateľ môže rozhodnúť, že či ide o pre neho pochopiteľnú akciu (a že či nieje priradená nová akcia pod týmto istým ID)
- „additional_data”
  - pokiaľ akcia obsahuje ďalšie informácie, tak sa pošlú v tomto políčku,
    - napríklad ak predpokladáme, že akcia je prepni kanál tak v „additional_data” môže byť uložené číslo kanálu, na ktorý treba prijímateľa správy prepnúť
    - je predpoklad, že prijímač bude rozumieť formátu prijatých dát. Protokol neurčuje, že čo má byť obsahom tohto atribútu.

Správy typu „forget“

- „GUID_event“ resp. „GUID_action“ - môžu sa vyskytovať viac krát

- identifikuje, ktorá akcia alebo udalosť v danej verzii už nie je podporovaná.
- toto je spôsob, ako informuje hlavná aplikácia klientskú, že nevie vykonať, spracovať udalosť.

Správa typu „request ping“

Ide o správu, ktorej odpoveďou je správa typu „ping“ na overenie, či niektorý z komunikujúcich náhle neprestal počúvať. Tieto správy sú potrebné, len ak medzi zariadeniami dlhší čas nenastala žiadna komunikácia. Protokol úmyselne nedefinuje ako často má dať o sebe klientská aplikácia vedieť a ponecháva toto rozhodnutie na hlavnú aplikáciu.

Správy typu „ping“

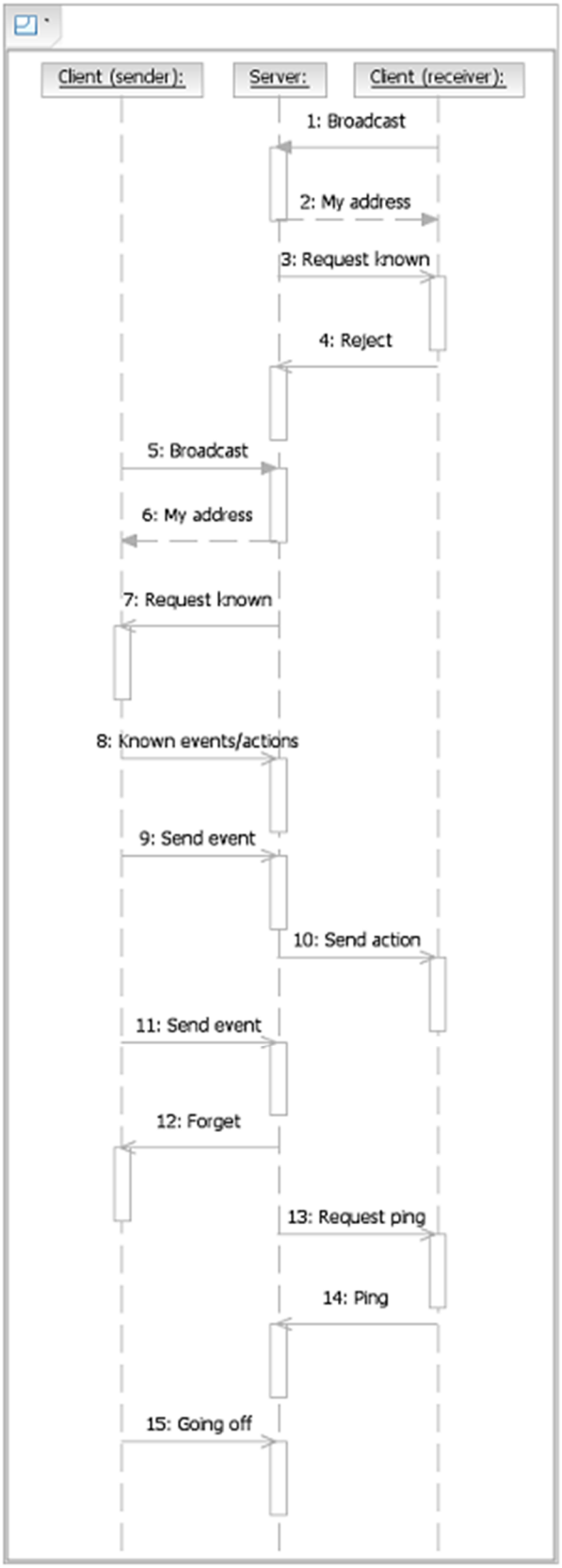
Ide o jednoduchú správu definujúcu, že aplikácia je stále pripojená a počúva. Táto správa by mala byť čo najrýchlejšie odoslaná po prijatí správy „request ping“

Správa typu „going off“

Ide o informačnú správu, ktorá informuje o tom, že odosielateľ správy práve prestáva počúvať na príchodzie správy.

3.2.4 Komunikácia medzi zariadeniami

3.2.4.1 Diagram komunikácie



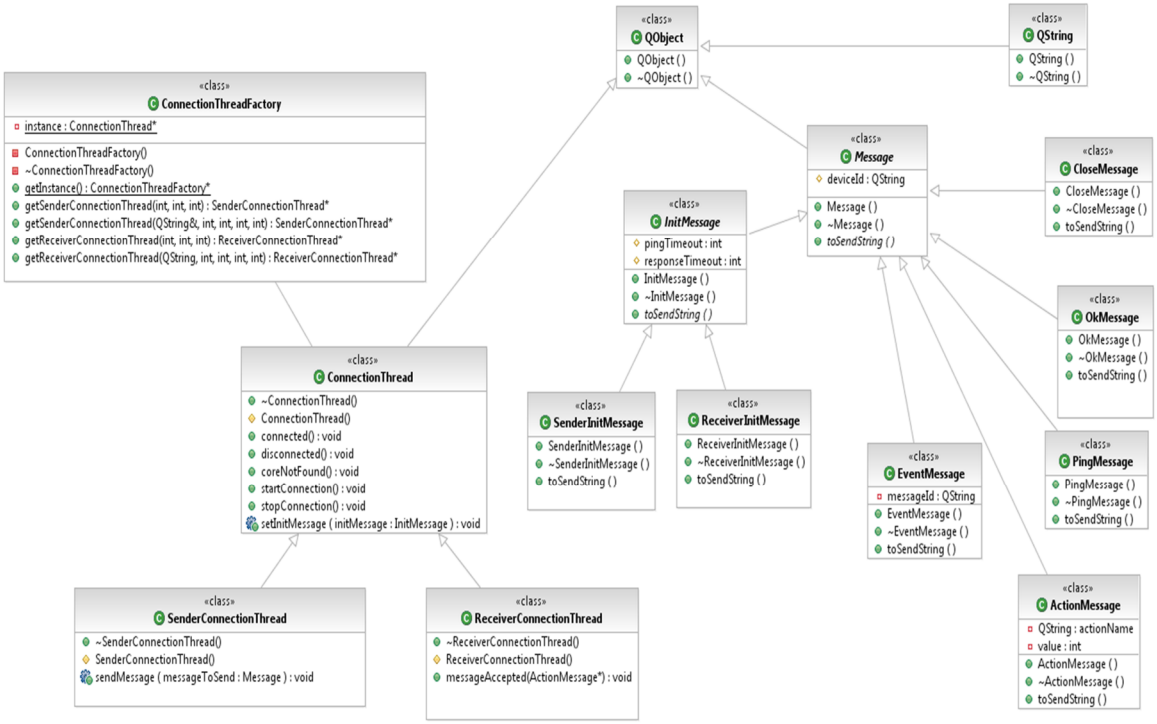
Obrázok 3-3: Diagram komunikácie klienta a servera

3.3 Implementácia komunikačného protokolu v jazyku C++

Keďže vyvíjaný systém bude rozdelený do modulov, ktorých komunikácia bude prebiehať cez sieťové pripojenie, vznikla potreba implementovať navrhnutý protokol. Je možné, že rôzne moduly budú implementované v rôznych jazykoch a preto aj tento protokol bude implementovaný pre jazyky C++ a Java.

3.3.1 Implementácia komunikačného protokolu v jazyku C++

Táto časť dokumentácie popisuje implementáciu protokolu v jazyku C++.



Obrázok 3-4: Diagram tried

Návrh knižnice najlepšie objasňuje Obrázok 3-4. Pre implementáciu komunikačnej knižnice v C++ bude použitý QT rámec. Nami navrhnutá hierarchia tried dedí od triedy *QtObject*, ktorá je poskytovaná QT rámcom.

ConnectionThread je naša vlastná abstraktná trieda reprezentujúca spojenie. Má chránený konštruktor a z toho dôvodu je objekt tejto triedy vytváraný v ConnectionThreadFactory. Tento postup sme zvolil z dôvodu neskoršieho jednoduchého doplnenia o nové parametre pri vytváraní ConnectionThread objektu.

SenderConnectionThread predstavuje triedu, ktorá zabezpečuje posielanie udalostí do centrálnej aplikácie. To znamená, že ak používateľ vykoná nejakú udalosť, napríklad

gesto, ako kliknutie na tlačidlo alebo hlasový vstup, táto trieda vyšle informáciu o danej udalosti.

SenderConnectionThread implementuje slot (metódu)

- *sendMessage(Message message)* - pošle objekt typu Message centrálnej aplikácii.

ReceiverConnectionThread predstavuje triedu pre prijatie akcie určenej na vykonanie v koncovom zariadení. Príkladom môže byť požiadavka na zvýšenie hlasitosti, táto akcia je poslaná z centrálnej aplikáciu na zariadenie, napríklad televízor.

ReceiverConnectionThread vysiela signál

- *messageAccepted(ActionMessage actionMessage)* - je vyvolaný na príchod akcie z centrálnej aplikácie.

SenderConnectionThread a ReceiverConnectionThread vysielajú signály a metódy zdedené od ConnectionThread

- *startConnection()* - inicializuje spojenie s cieľom nadviazať komunikáciu s centrálnou aplikáciou,
- *stopConnection()* - ukončuje spojenie s centrálnou aplikáciou,
- *connected()* - signál je vyslaný vláknom, ak dôjde ku korektnému spojeniu s hlavnou aplikáciou,
- *disconnected()* - signál je vyslaný vláknom, ak dôjde k odpojeniu od hlavnej aplikácie,
- *coreNotFund()* - signál vyslaný v prípade, že pokus o nájdenie centrálnej aplikácie nebol úspešný.

Ako už bolo spomenuté, vytvorenie ConnectionThread objektu, či už pre prijímanie alebo odosielanie, prebieha v triede ConnectionThreadFactory. Táto trieda je implementovaná ako SingletonPattern architektonický vzor, aby sa zamedzilo zbytočnému vytváraniu nepotrebných objektov. To znamená, že počas behu aplikácie existuje vždy iba jeden objekt ConnectionThreadFactory, ktorý vytvára ľubovoľný počet objektov ConnectionThread.

ConnectionThreadFactory implementuje nasledujúce metódy:

- *getInstance()* – vracia smerník aktuálneho objektu ConnectionThreadFactory,

- *getSenderConnectionThread(int pingTimeout, int responseTimeou, int reconectTimetout)* – vytvorí objekt odosielateľa SenderConnectionThreads nastaveným časom čakania na odpoveď (responseTimeout), časom odosielania správ na udržanie spojenia (pingTimeout) a intervalom na opätovné pripojenie pri zlyhaní spojenia; všetky časy sú uvádzané v milisekundách; na nájdenie centrálnej aplikácie sa použije UDP broadcast s nami definovaným formátom správy,
- *getSenderConnectionThread(QString ipAddress, int port, int pingTimeout, int responseTimeout, int reconectTimetout)* – na rozdiel od predošlej metódy sa na pripojenie k centrálnej aplikácii použije jej IP adresa a port,
- *getReceiverConnectionThread(int pingTimeout, int responseTimeout, int reconectTimetout)* – vytvorí objekt príjímateľa ReceiverConnectionThread s nastavením časom čakania na odpoveď (responseTimeout), časom odosielania správ na udržanie spojenia (pingTimeout) a intervalom na opätovné pripojenie pri zlyhaní spojenia; všetky časy sú uvádzané v milisekundách; na nájdenie centrálnej aplikácie sa použije UDP broadcast s nami definovaným formátom správy,
- *getReceiverConnectionThread(QString ipAddress, int port, int pingTimeout, int responseTimeout, int reconectTimetout)* – v tomto prípade sa vytvorí spojenie s centrálnou aplikáciou s použitím jej IP adresy a portu.

Počas sieťovej komunikácie budú posielané správy vo formáte definovanom naším connection-kin-transport-protorokolom (CKTP). Pre reprezentovanie správy použijeme vlastnú triedu Message, ktorá je abstraktná. Od nej dedia ďalšie triedy, ktoré reprezentujú rôzne druhy správ:

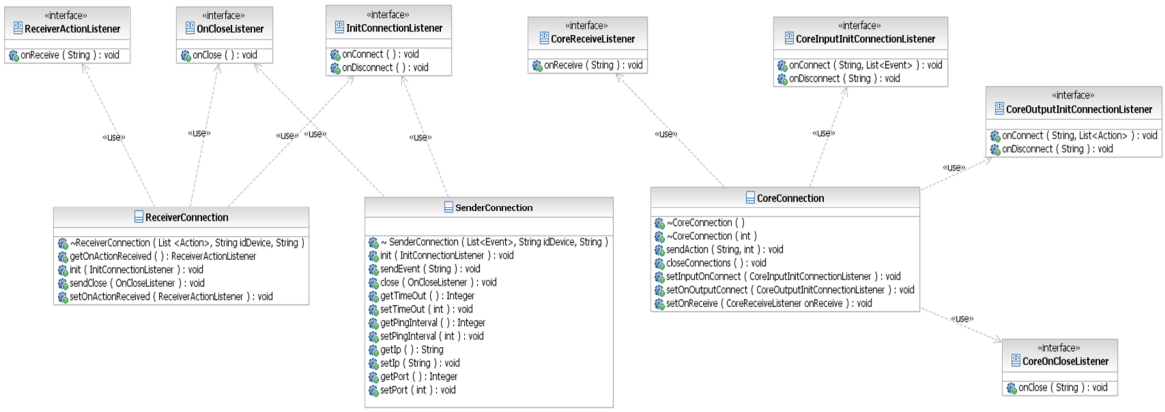
- InitMessage (len abstraktná),
- SenderInitMessage,
- ReceiverInitMessage,
- EventMessage,
- ActionMessage,
- PingMessage,
- OkMessage,
- CloseMessage.

Každá z týchto správ implementuje metódu

- *toSendString()* – vracia správu spolu s hlavičkou v podobe pripravenej na odoslanie.

3.4 Implementácia komunikačného protokolu v jazyku Java

Táto časť dokumentácie popisuje implementáciu protokolu v jazyku Java (Obrázok 3-5: Java protokolObrázok 3-5).



Obrázok 3-5: Java protokol

Implementácia komunikačnej knižnice v jazyku Java je rozdelená do troch tried:

- *SenderConnection*
- *ReceiverConnection*
- *CoreConnection*

SenderConnection predstavuje triedu, ktorá zabezpečuje posielanie udalostí do centrálnej aplikácie. Ak používateľ vykoná nejakú udalosť, napríklad gesto, kliknutie na tlačidlo alebo hlasový vstup, táto trieda vyšle informáciu o danej udalosti.

SenderConnection implementuje metódy:

- *SenderConnection(List<Event> events, String deviceName, String idDevice) { }* – pošle centrálnej aplikácií zoznam všetkých definovaných udalostí, meno zariadenia a jeho identifikačné číslo
- *init(InitConnectionListener callback) { }* – nadviaže spojenie s centrálnou aplikáciou
- *sendEvent(final String eventId) { }* – odošle centrálnej aplikácii informáciu o udalosti, ktorá nastala

- *close(final OnCloseListener onClose) { }* – ukončí spojenie s hlavnou aplikáciou

ReceiverConnection predstavuje triedu pre prijatie akcie určenej na vykonanie na koncovom zariadení. Príkladom môže byť požiadavka na zvýšenie hlasitosti. Táto akcia je poslaná z centrálnej aplikácie na želané koncové zariadenie, napríklad televízor.

ReceiverConnection implementuje metódy:

- *ReceiverConnection(List<Action> actions, String deviceName,String idDevice) { }* - pošle centrálnej aplikácií zoznam všetkých definovaných akcií, meno zariadenia a jeho identifikačné číslo
- *init(InitConnectionListener callback) { }* – nadviaže spojenie s centrálnou aplikáciou
- *sendClose(final OnCloseListener onClose) throws IOException { }* –pokúsi sa ukončiť spojenie s hlavnou aplikáciou
- *ReceiverActionListener getOnActionReceived() { }* – metóda slúžiaca na príjem akcie zo strany centrálnej aplikácie
- *setOnActionReceived(ReceiverActionListener onActionReceived) { }* – zabezpečí informovanie zariadenia o prijatí akcie

CoreConnection predstavuje triedu slúžiacu na prijímanie udalostí od odosielajúcich zariadení a odosielanie správ

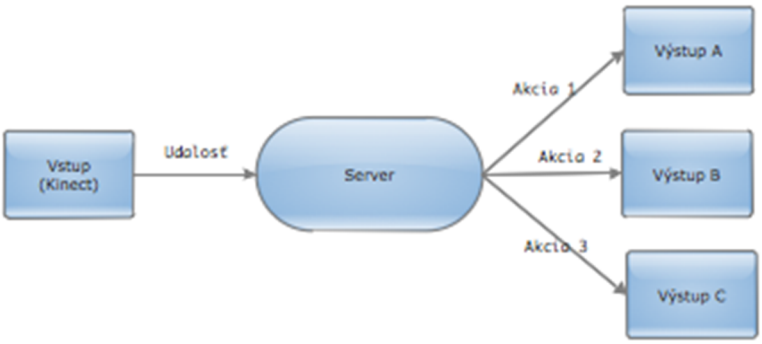
- *CoreConnection() throws IOException { }* – nadviazanie spojenia so zariadením
- *CoreConnection(int port) throws IOException { }* – nadviazanie spojenia na príslušnom porte
- *sendAction(String deviceId, String actionId, int value) { }* – odoslanie akcie prijímaču
- *closeConnections() { }* – ukončenie spojenia
- *setInputOnConnect(CoreInputInitConnectionListener onConnect) { }* – komunikácia s odosielajúcim zariadením
- *setOnOutputConnect(CoreOutputInitConnectionListener onConnect) { }*- komunikácia s prijímacím zariadením
- *setOnReceive(CoreReceiveListener onReceive) { }* - metóda zodpovedná za príjem udalosti od odosielacieho zariadenia

3.5 Mapovanie udalostí na akcie

Jeden z hlavných prínosov projektu FIIT Kinect má byť možnosť priradenia rôznych udalostí (t.j. gest) ku povelom pre rôzne zariadenia. Používateľ si tak môže určiť ktorými udalosťami bude ovládať ktoré zariadenia.

3.5.1 Návrh

Z dôvodu obsluhovania viacerých zariadení jednou aplikáciou je potrebné zadefinovať mapovanie udalostí (napr. gest) na zariadenia a akcie, ktoré sa majú vykonať. Musí byť určené ako sa budú odlišovať akcie pre jednotlivé zariadenia. Pre lepšie predstavenie problému je uvedený Obrázok 3-6.

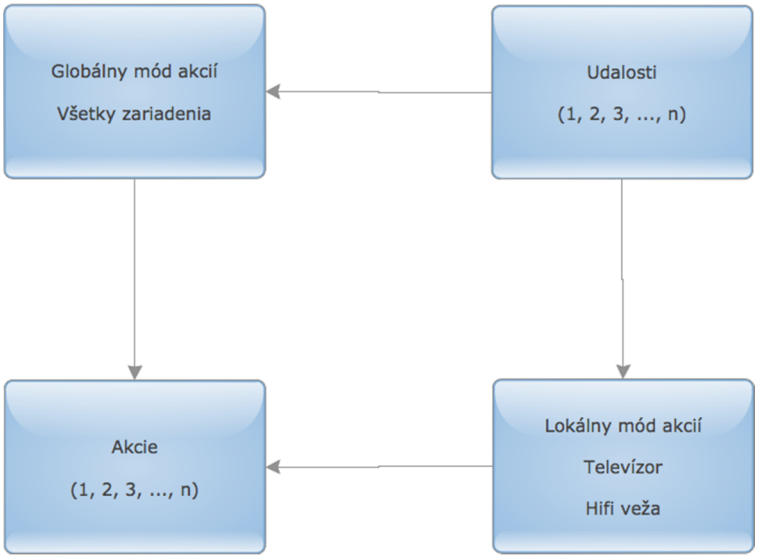


Obrázok 3-6: Vzťah udalostí a akcií

Bolo potrebné vybrať z možností mapovania

- 1:1,
- 1:n,
- n:n.

Výber mal zohľadniť obmedzené množstvo vstupných udalostí a zároveň zabezpečiť jednoznačnosť správania aplikácie. Bolo vybrané riešenie, ktoré obsahuje 2 primárne módy správania sa aplikácie reprezentované Obrázok 3-7.



Obrázok 3-7: Globálny a lokálny mód

Globálny mód akcií je zapnutý automaticky pri zapnutí aplikácie. Používateľ v tomto móde môže použiť všetky udalosti, ktoré sú nastavené pre tento mód. Udalosti obsluhujúce akcie z globálneho módu sú aktívne aj v momente, ak je používateľ prepnutý do lokálneho módu. Nie je možné použitie udalostí namapovaných v globálnom móde v rámci lokálnych módov.

Lokálny mód akcií je skupina akcií definovaných len pre určité výstupné zariadenie, resp. skupinu zariadení. Lokálny mód akcií sa zapína špeciálnymi preddefinovanovanými udalosťami určenými na prepínanie medzi módmi (udalostí môže viac, na prepnutie stačí jedna z nich). Tieto akcie sú unikátne určené len na prepínanie medzi módmi aplikácie. Na prepínanie medzi módmi je možné používať len globálne udalosti.

Dvojicu udalosť - akcia, ktorá vznikla v procese mapovania je možné pre lepšiu orientáciu nazvať situáciou. Situácia je jednoznačné identifikovaná pomocou udalosti, ktorou je vyvolaná a akciou, ktorú spúšťa. Existujú 2 typy situácií štandardné (spúšťajú akciu) a prepínacie (existujú len v rámci globálneho módu a slúžia na prepínanie medzi módami).

4 Šprint 2 (Ružový panter)

4.1 Návrh GUI

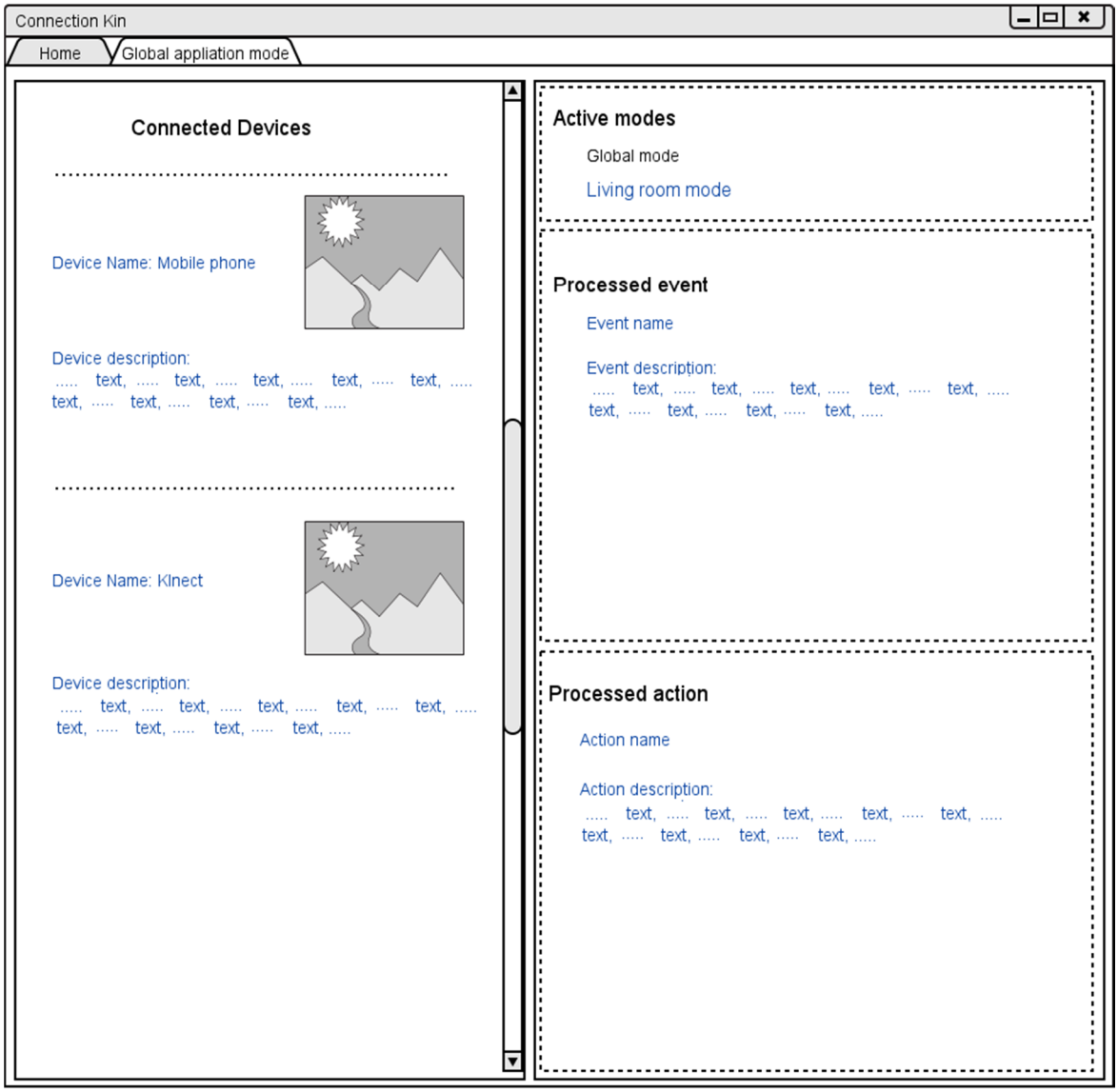
4.1.1 Úvodná obrazovka

Úvodná obrazovka centrálnej aplikácie poskytuje používateľovi prehľad o aktuálnom stave aplikácie, pričom informácie sa aktualizujú v reálnom čase.

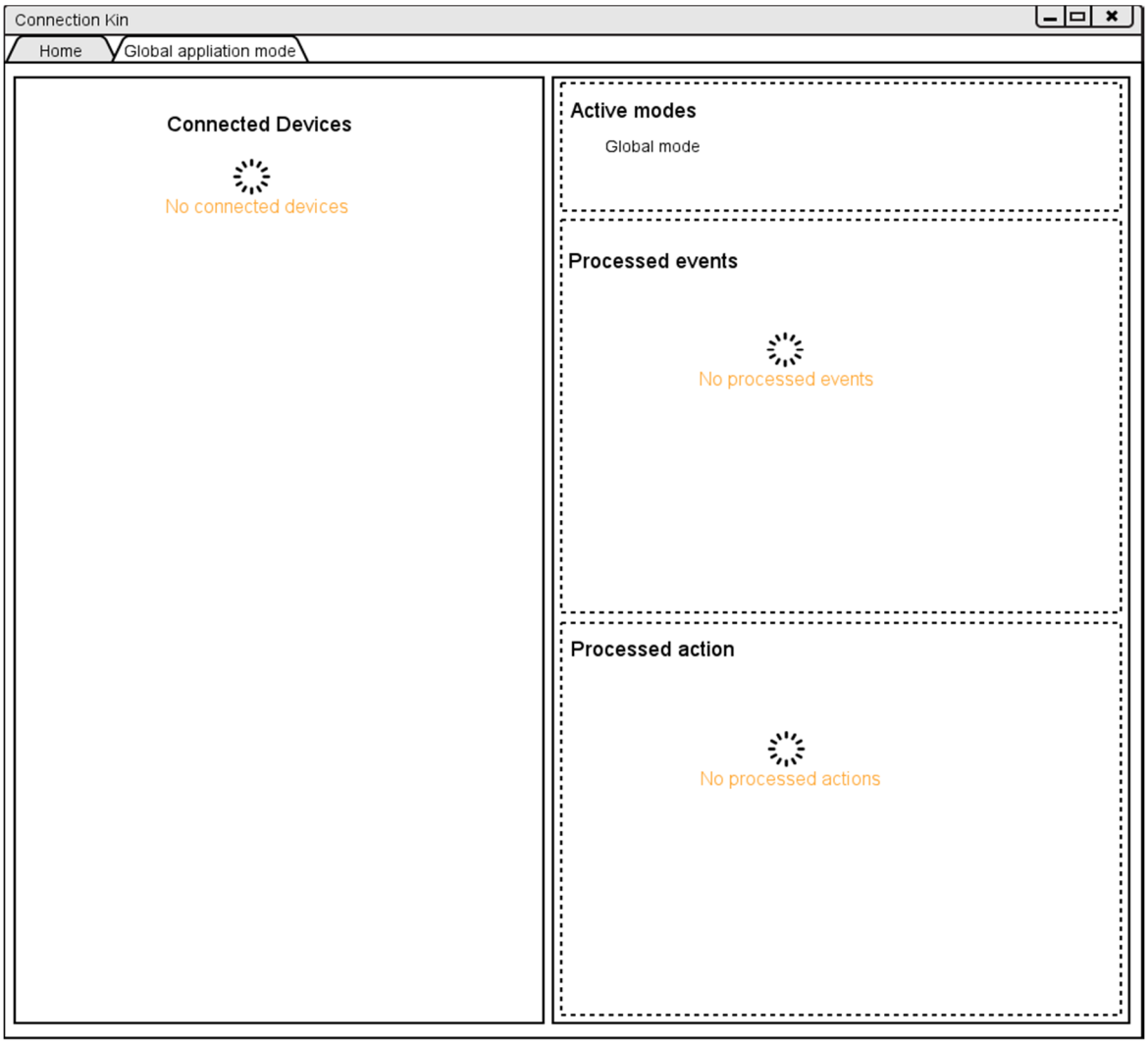
Vľavo sú zobrazené pripojené zariadenia a ich charakteristika doplnená obrázkom. Napravo sú uvedené aktuálne aktívne módy. Napravo sa taktiež zobrazuje informácia o tom aká udalosť nastala a aká akcia bola vyvolaná, spolu s ich názvom a popisom (Obrázok 4-1).

Z úvodnej obrazovky je možnosť presunúť sa do ďalšej tabu, respektíve záložky (tab-u) globálneho módu aplikácie.

4 | Šprint 2 (Ružový panter)



Obrázok 4-1: Úvodná obrazovka



Obrázok 4-2: Prázdna úvodná obrazovka

4.1.2 Aplikačný globálny mód

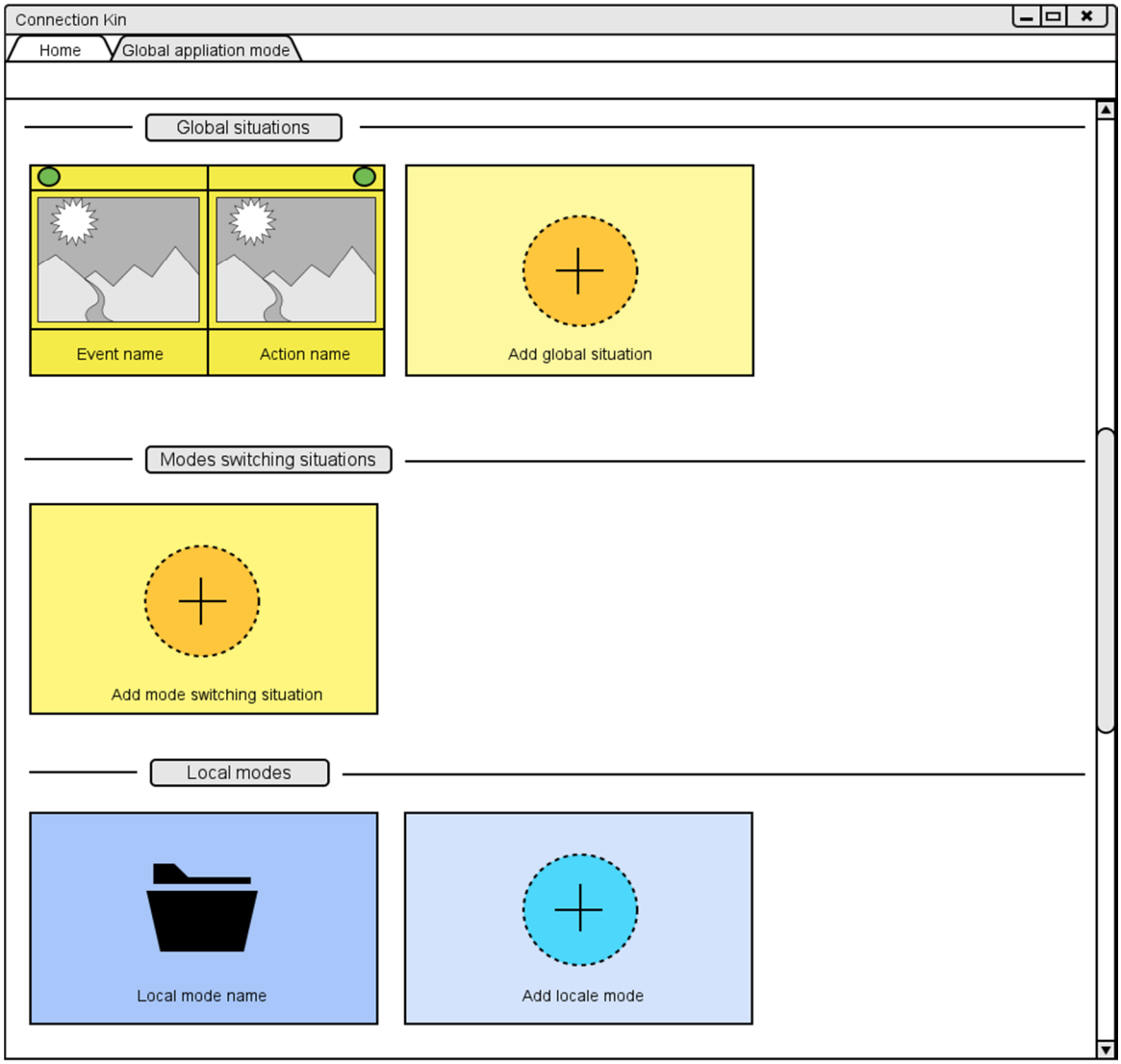
Vo forme priečinkov sú organizované a zobrazené jednotlivé nastavenia centrálnej aplikácie (Obrázok 4-3).

Situáciu chápeme ako jedno konkrétne mapovanie udalosti a akcie. Na obrazovke sú jednotlivé nadefinované situácie globálneho módu, a taktiež tlačidlo pre vytvorenie novej situácie. Situácia je zobrazená ako dvojica obrázkov a názvov pre udalosť a akciu. Nad obrázkom udalosti aj akcie sa nachádza príznak - farebný kruh, ktorý indikuje či je možné situáciu vykonať, teda či sú pripojené požadované zariadenia.

Samostatne oddelené sú situácie na prepínanie medzi módmi. Ich zobrazenie je založené na rovnakom princípe ako zobrazenie ostatných globálnych situácií. Akcie na prepínanie módov sú akcie poskytnuté aj vyvolávané priamo v centrálnej aplikácii. Používateľ k nim

môže priradiť ľubovoľnú udalosť, a to po stlačení tlačidla pre vytvorenie novej situácie na prepínanie módov.

Globálny mód aplikácie okrem situácií definuje ďalšie lokálne módy. Lokálne módy sú taktiež usporiadané v podobe priečinkov. Tlačidlom možno pridávať nové lokálne módy.



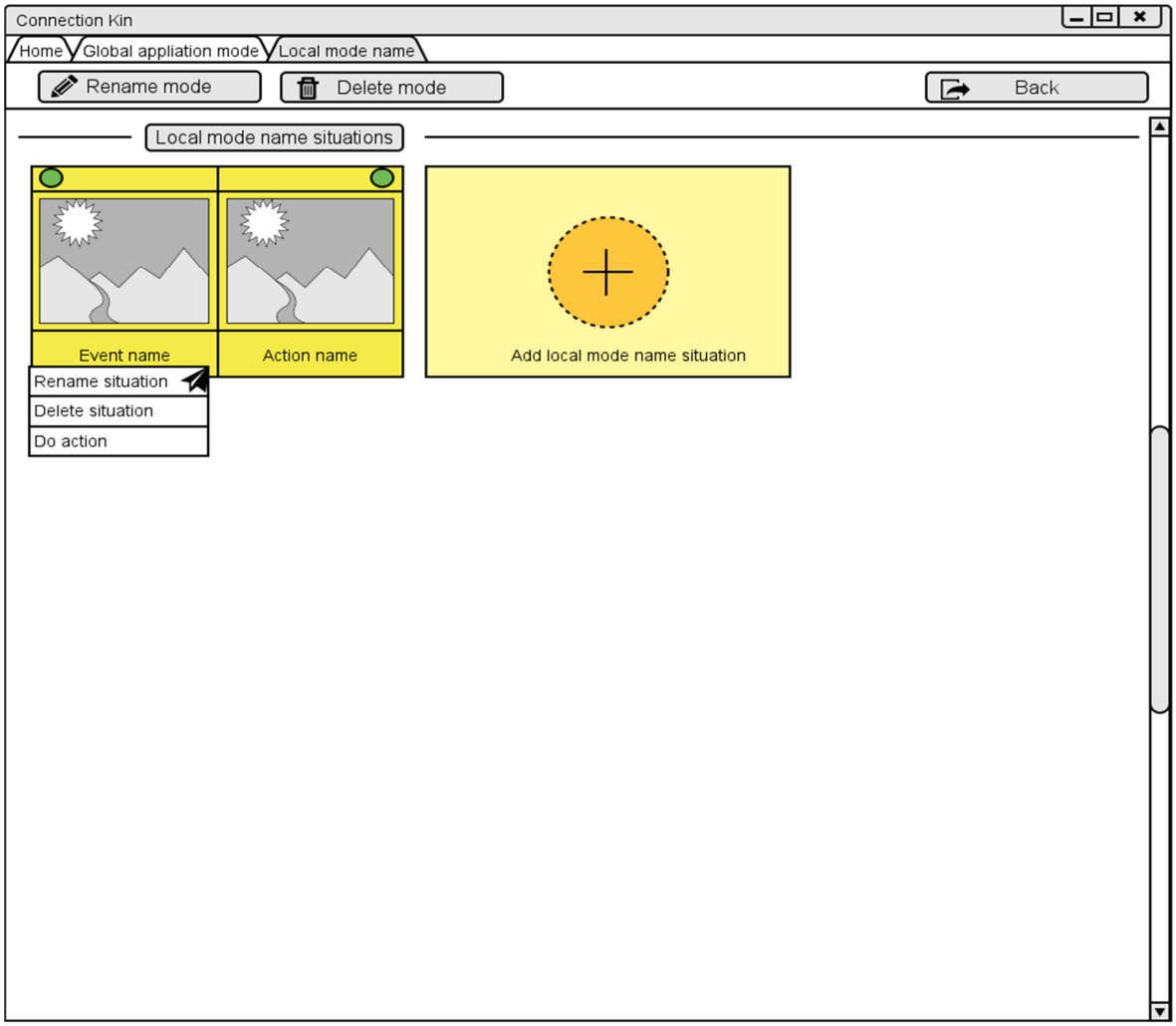
Obrázok 4-3: Aplikačný globálny mód

4.1.3 Lokálny mód

Ak používateľ v globálnom móde dva krát klikne na konkrétny lokálny mód, otvorí sa nová záložka s definovaným lokálnym módom, ktorá nebola doposiaľ k dispozícii (Obrázok 4-4). Používateľ je do záložky automaticky prepnutý. Táto záložka nie je viditeľná v základnej ponuke po otvorení aplikácie, a to z dôvodu zachovania prehľadnosti a štruktúrovanosti nastavení aplikácie.

V zobrazení lokálneho módu sú k dispozícii voľby pre premenovanie a mazanie lokálneho módu. Podobne ako v globálnom móde sú tu zobrazené jednotlivé situácie a aj možnosť pridania novej situácie lokálneho módu.

Po kliknutí pravým tlačidlom myši na niektorú zobrazenú situáciu sa vyroluje menu s ponukou aktivít, ktoré možno vyvolať nad situáciou. Vybraná voľba sa vykoná.



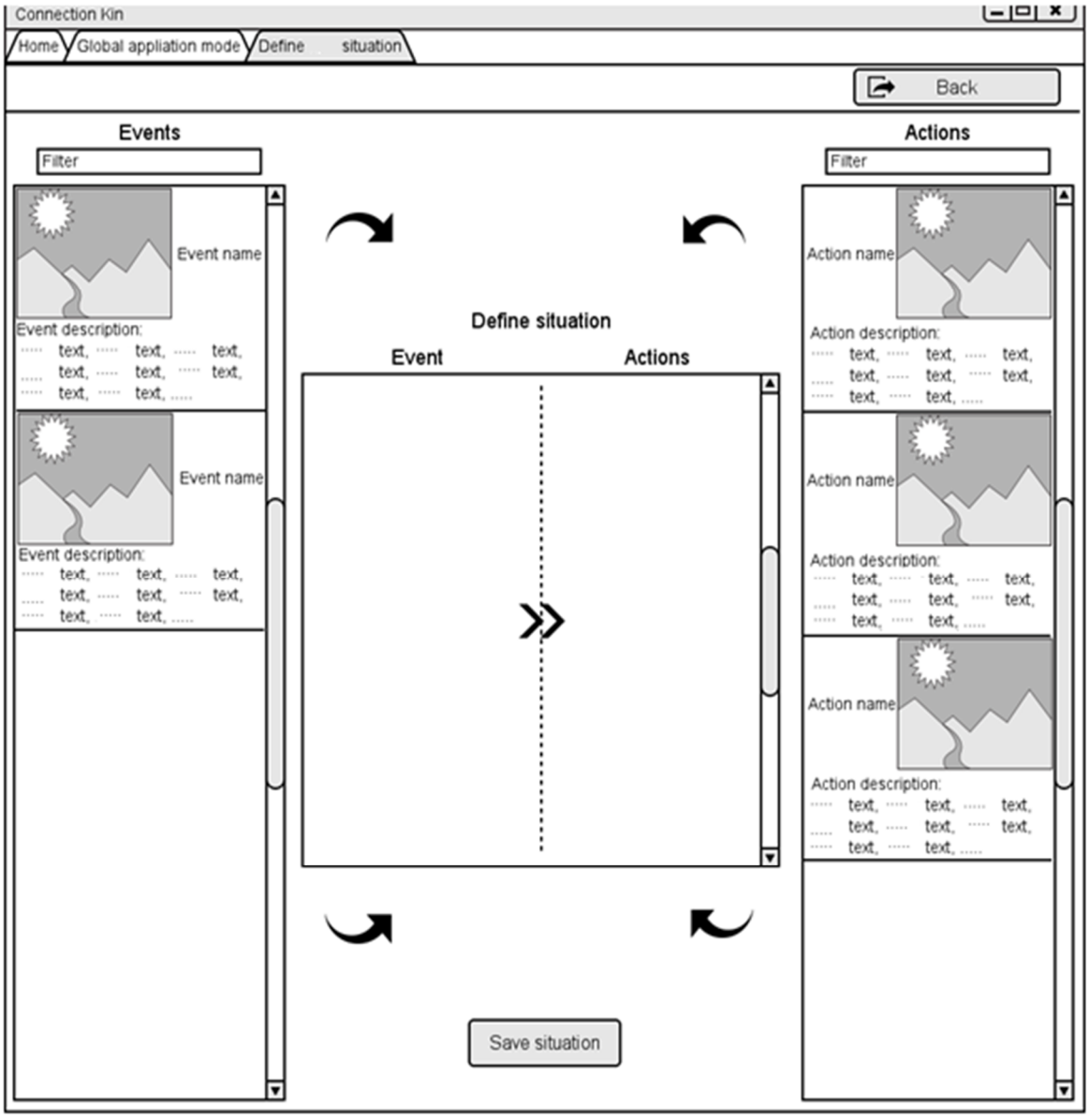
Obrázok 4-4: Lokálny mód

4.1.4 Definovanie novej situácie

Novú situáciu, ktorú chce používateľ pridať, je nutné najskôr definovať. S týmto účelom sa otvorí ďalšia nová záložka pre definovanie situácie (Obrázok 4-5).

Vľavo sa nachádza ponuka udalostí a vpravo je ponuka akcií. Ponuky možno filtrovať na základe zadaného textu.

Do prostredného priestoru môže používateľ technikou drag-and-drop vkladať udalosti a akcie, ktoré chce použiť v situácii.



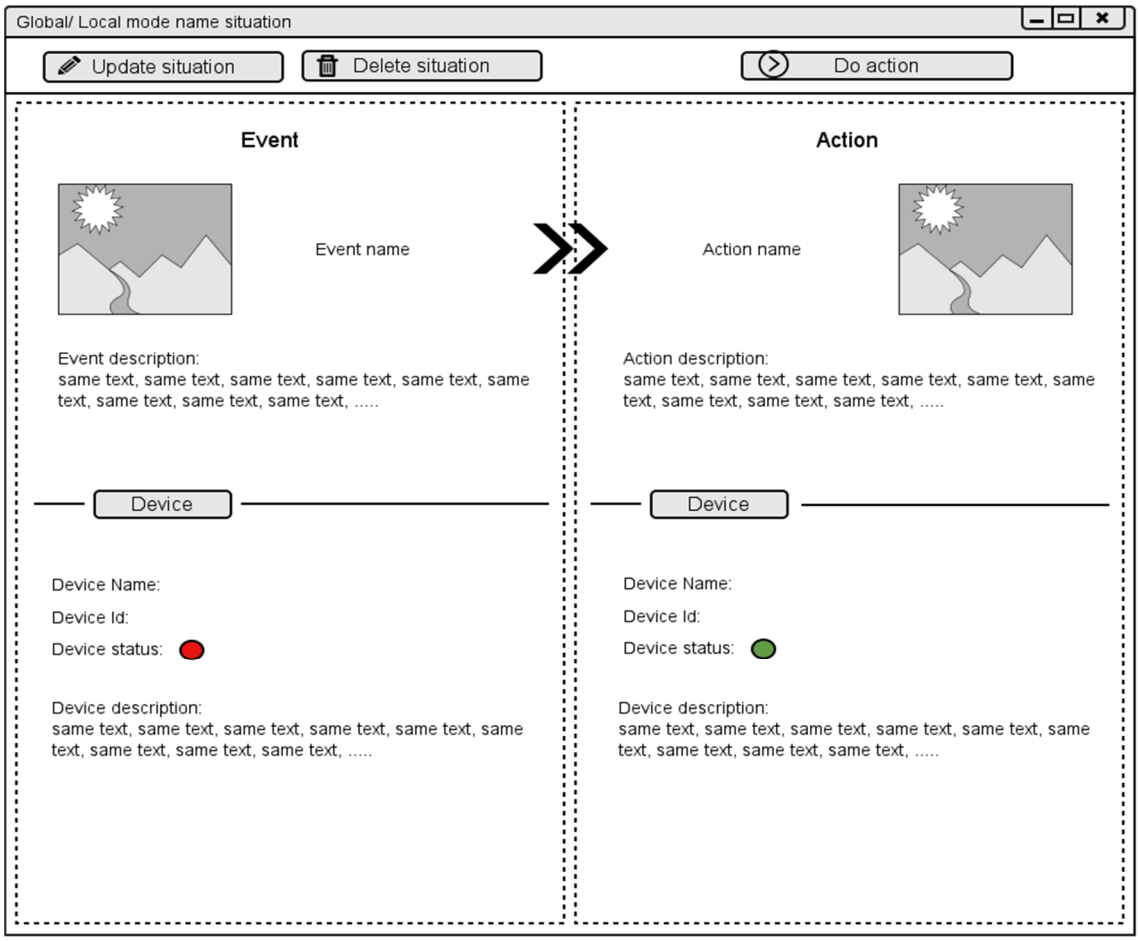
Obrázok 4-5: Definovanie novej situácie

4.1.5 Detail situácie

V prípade, že chce používateľ vidieť detail situácie, definovanej či už v globálnom alebo lokálnom móde, má možnosť otvoriť okno s detailom situácie (Obrázok 4-6), a to dvojklikom na vybranú situáciu.

K dispozíciu má obrázok, názov, popis ako aj charakteristiku zariadení, na ktorých sú udalosti a akcie vykonávané. Používateľ môže vykonať uvedenú akciu aj priamo v centrálnej aplikácii kliknutím na príslušné tlačidlo.

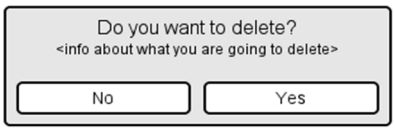
Zobrazenú situáciu možno zmazať alebo upraviť. Po stlačení voľby na úpravu sa otvorí obrazovka pre definovanie situácie. Táto obrazovka zodpovedá aktuálnemu nastaveniu situácie, ale s možnosťou zmeny týchto nastavení.



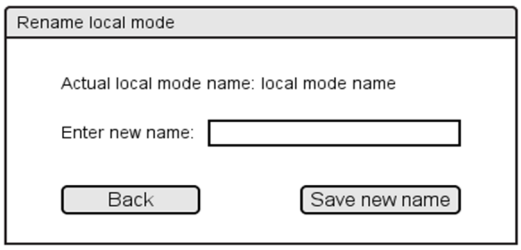
Obrázok 4-6: Detail situácie

4.1.6 Pomocné dialógové okná

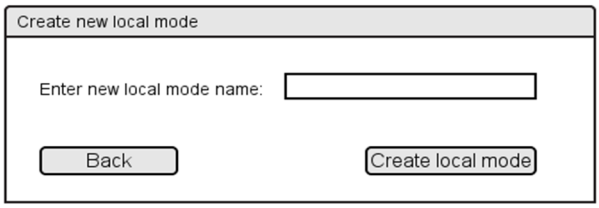
Okrem hlavných obrazoviek sú pripravené pomocné dialógové okná pre potvrdzovanie vymazania situácie a lokálneho módu (Obrázok 4-7). Dialógové okno je vytvorené aj pre prípad premenovania lokálneho módu (Obrázok 4-8) alebo vytvorenia nového lokálneho módu (Obrázok 4-9).



Obrázok 4-7: Potvrdzovacie okno



Obrázok 4-8: Premenovanie lokálneho módu

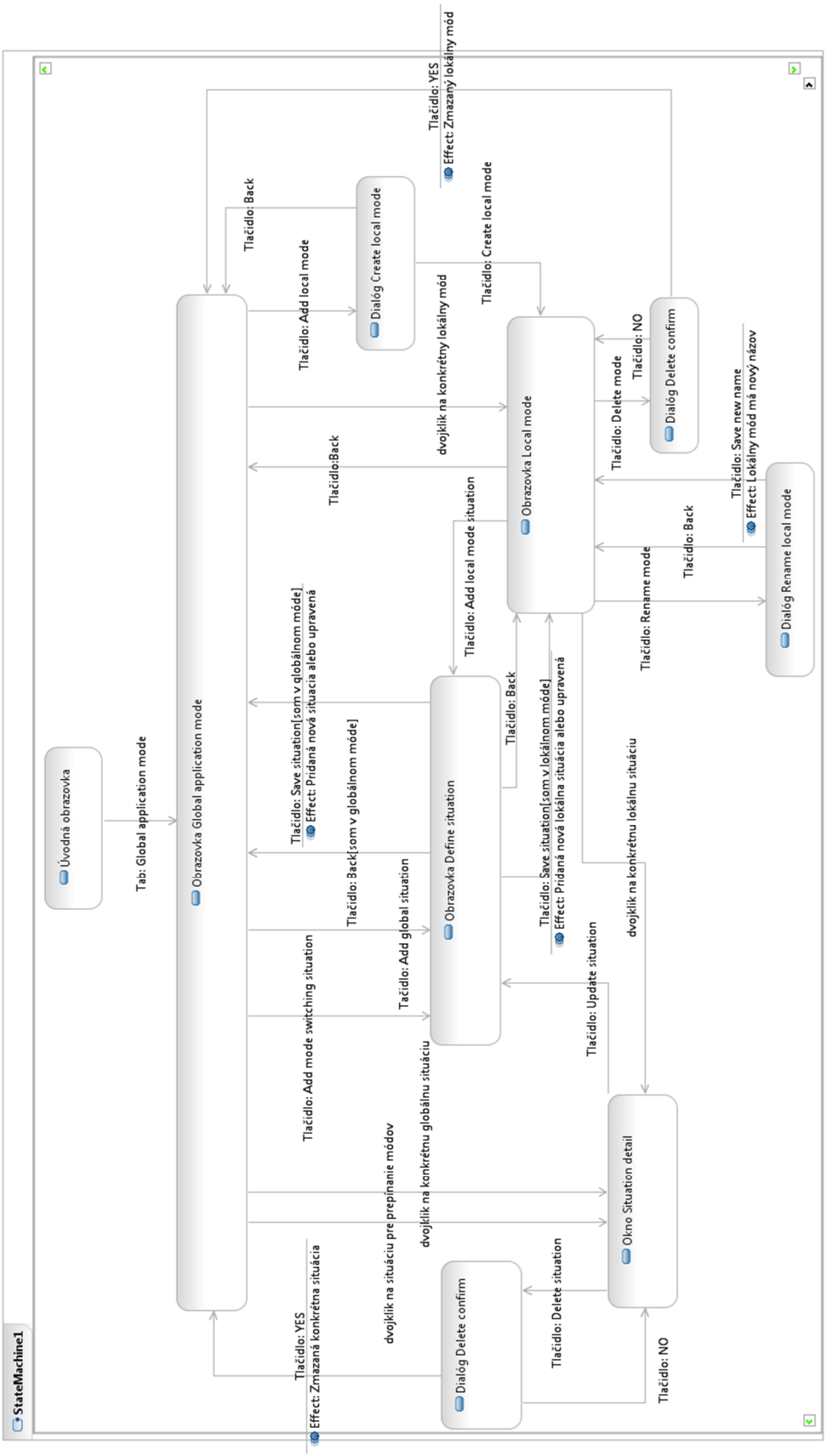


Obrázok 4-9: Vytvorenie nového lokálneho módu

4.1.7 Prepínanie medzi obrazovkami

Jednotlivé obrazovky a okná je možné prepínať na základe voľby používateľa. Vzájomná interakcia prepínania obrazoviek je vyjadrená v stavovom diagrame na obrázku (Obrázok 4-10). Používateľské rozhranie je navrhnuté tak, aby mal používateľ vždy príležitosť vrátiť sa späť, ako aj vymazať a upraviť nadefinované možnosti.

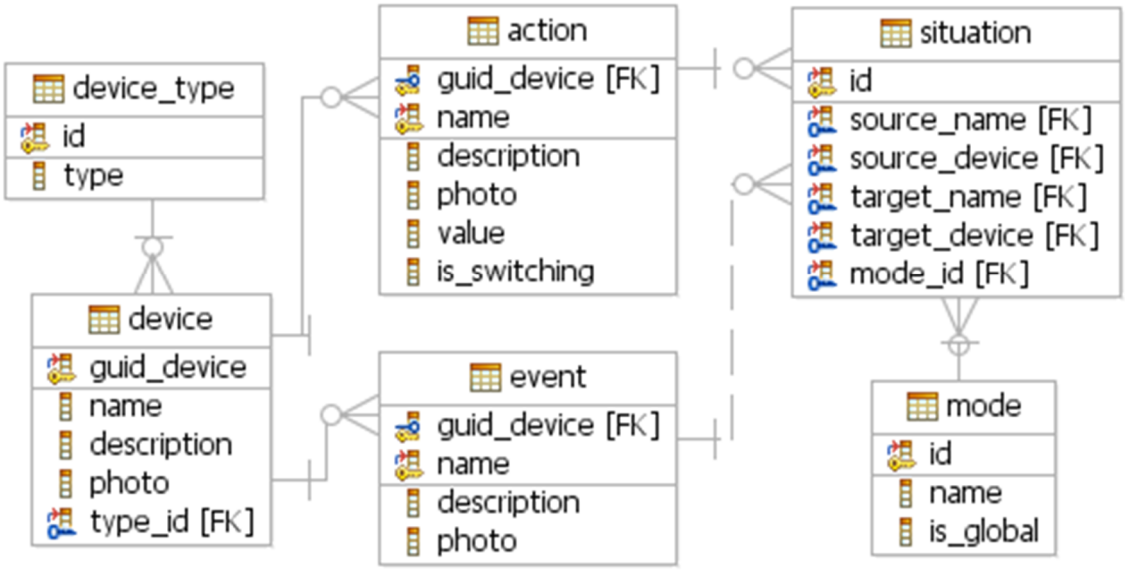
4 | Šprint 2 (Ružový panter)



Obrázok 4-10: Stavový diagram prepínania obrazoviek

4.2 Návrh databázy

Na nižšie uvedenom obrázku (Obrázok 4-11) sa nachádza dátový model spoločne navrhnutej databázy. Obsahom tejto kapitoly je aj opis modelu.



Obrázok 4-11: Dátový model

Základná tabuľka je device, ktorá obsahuje záznam o každom zariadení, ktoré sa k aplikácii úspešne pripojilo. Táto tabuľka obsahuje polia *guid_device* pre jednoznačnú identifikáciu zariadenia, *name* teda názov zobrazovaný používateľovi, ďalej *description* pre prípadný bližší popis zariadenia, *photo* k vizuálnej identifikácii a taktiež *type_id* pre určenie typu zariadenia.
Typ zariadenia je cudzí kľúč k tabuľke *device_type*. Tá určuje, či sa jedná o zariadenie typu prijímač alebo vysielač.

Ku konkrétnym zariadeniam sa taktiež viaže tabuľka event, ktorá definuje udalosť, ktorá môže na zariadení nastať. Každá udalosť je jednoznačne identifikovaná podľa zariadenia, na ktorom nastala a názvu udalosti. Tabuľka *event* je naviazaná len na vysielajúce zariadenia. Tabuľka okrem stĺpcov *guid_device* a *name* obsahuje stĺpce *description* pre popis udalosti a *photo* pre možnú vizuálnu identifikáciu udalosti.

Obdobne pre zariadenia slúžiace ako prijímače je tu tabuľka *action*, ktorá definuje akcie, ktoré sa majú na koncovom zariadení vykonať. Tabuľka obsahuje všetky stĺpce tabuľky event, a okrem nich aj stĺpce *value* a *is_switching*. Stĺpec *value* definuje hodnotu, o ktorú sa má daná akcia vykonať, resp. koľko krát. Napríklad môže obsahovať akciu s názvom stlmiť s hodnotou 0, ktorá môže byť vnútorne reprezentovaná ako stíšiť hlasitosť na

hodnotu 0. Pole *is_switching* hovorí o tom, či ide o situáciu, ktorá mení aktuálny mód rozpoznávania gest na nejaký iný.

Samotná aplikácia ale musí taktiež vedieť, že ktorú akciu má vykonať keď nastane konkrétna udalosť. K tomu slúži väzobná tabuľka *situation* označujúca takúto situáciu. Táto tabuľka obsahuje svoj identifikátor ako primárny kľúč, riadky *source_name* a *source_device* pre jednoznačnú identifikáciu udalosti a obdobne *target_name* a *target_device* pre unikátnu identifikáciu akcie. Ďalej obsahuje pole *mode*, ktoré určuje o aký mód ide. V tomto prípade ide o referenciu na tabuľku *mode*.

Naviazaná tabuľka *mode* obsahuje len 3 stĺpce. Prvý je primárny kľúč, na ktorý sa naväzuje tabuľka *situation*, druhý je názov daného módu a posledný sa volá *is_global* a ten určuje to, či ide o globálny mód, ktorého situácie sú dostupné stále.

4.3 Návrh centrálnej aplikácie

Ako bolo spomenuté, projekt je rozdelený na niekoľko modulov, samostatných aplikácií, ktoré komunikujú po sieti. Jedným z týchto modulov je centrálna aplikácia. Táto aplikácia čaká na pripojenie rôznych externých aplikácií, ktoré spravujú koncové zariadenia. V tejto časti budeme zamieňať pojem zariadenia a externej aplikácie, ktorá ho spravuje. Takýmito zariadeniami môžu byť napr. Kinect, telefón alebo infračervený vysielač. Zariadenia rozdelíme na dve skupiny: vysielače a prijímače.

Zariadenia typu vysielač dostávajú vstup od používateľa a posielajú po sieti informácie centrálnej aplikácií. Môže to byť napr. hlasový povel, gesto alebo stlačenie tlačidla na telefóne. Pre takýto vstup budeme používať pomenovanie udalosť. Príkladmi zariadení typu vysielač sú Kinect alebo telefón.

Zariadenie typu prijímač očakáva príkazy od centrálnej aplikácie na vykonanie rôznych činností. Môže to byť napríklad zmena kanálu na televíznom prijímači, zmena hlasitosti hi-fi súpravy alebo zhasnutie svetla. Pre takúto činnosť budeme používať pojem akcia. Pre jednoduchšiu implementáciu obsahuje akcia aj číselnú hodnotu, ktorá ju bližšie definuje a má rôzny význam podľa akcie. Môže napríklad určiť, o koľko zvýšiť hlasitosť, alebo na ktorý kanál prepnúť. Príkladmi zariadení typu prijímač sú napríklad infračervený vysielač (ovládajúci televíziu, hi-fi) alebo aplikácia ovládajúca osvetlenie.

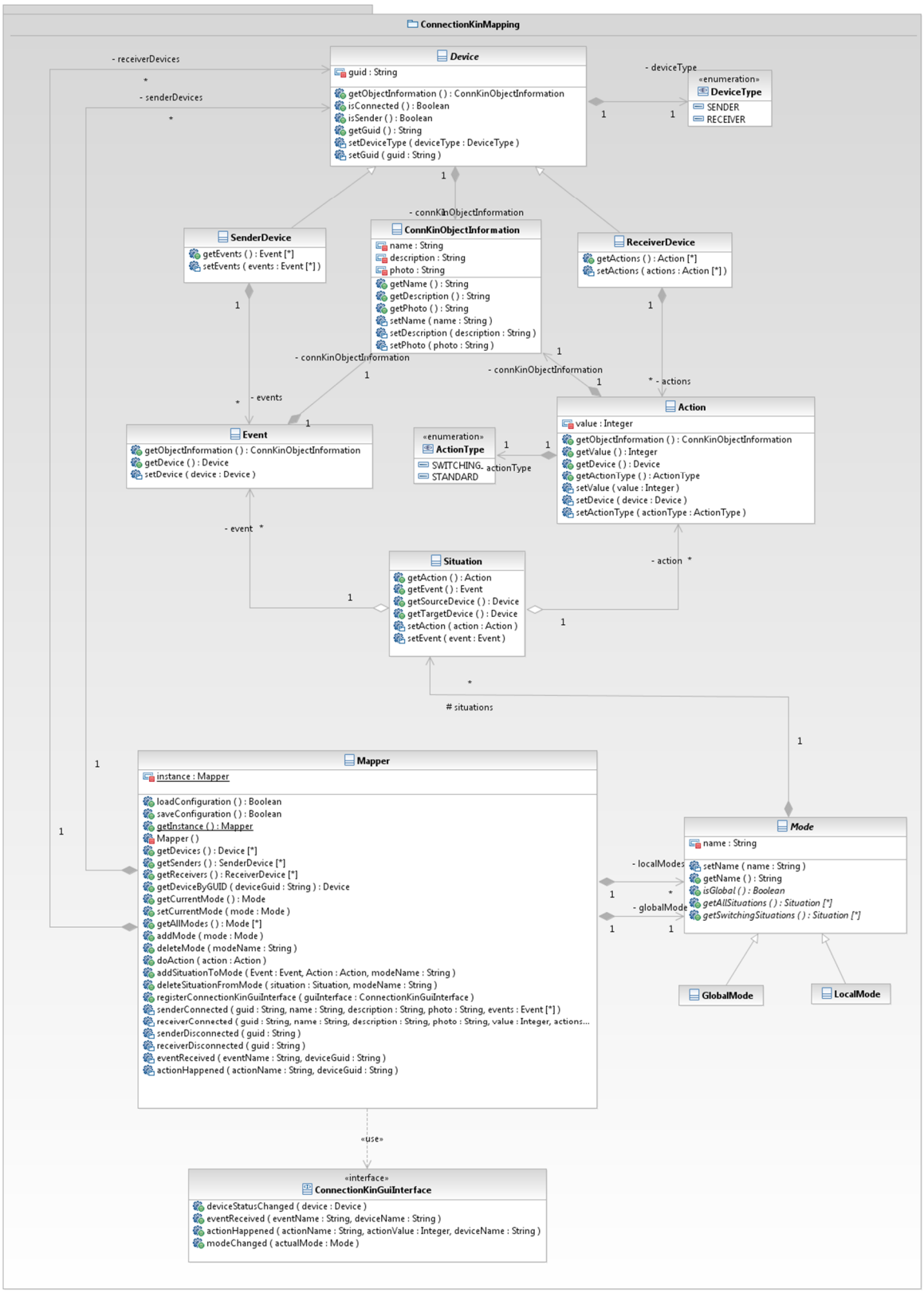
Centrálna aplikácia dovoľuje používateľovi definovať dvojice udalosť-akcia. Aplikácia po prijatí takejto udalosti pošle príkaz na vykonanie danej akcie. Používateľ tak má možnosť ovládať napr. televíziu pomocou gest, ktoré spracuje Kinect. Pre takéto dvojice udalosť-akcia budeme používať názov situácia.

Z dôvodu obmedzeného množstva udalostí sme sa rozhodli do aplikácie zaviesť pojem módu. Mód obsahuje situácie, ktoré môže používateľ pridávať, mazať a upravovať. V aplikácií budú dva typy módov: globálny a lokálny.

Globálny mód je vždy aktívny, tj. situácie, ktoré sú v globálnom móde sú vždy aktívne a ich akcie sa vykonajú po prijatí zodpovedajúcej udalosti. Globálny mód je v aplikácii práve jeden.

Lokálne módy si vytvára používateľ v centrálnej aplikácií. Lokálny mód musí byť aktivovaný udalosťou, ktorú určí používateľ. Aktívny môže byť naraz len jeden lokálny mód. Ak je mód aktívny, tak situácie sa vykonajú po prijatí zodpovedajúcej udalosti. Ak je mód neaktívny, tak aj keď centrálna aplikácia prijme udalosť, ktorá je priradená k niektorej situácii v takomto móde, akcia tejto situácie nebude vykonaná. Používateľ môže lokálne módy vytvárať, upravovať a mazať.

4 | Šprint 2 (Ružový panter)



Obrázok 4-12: Model mapovania

4.3.1 Mapper

Hlavná trieda, ktorá reprezentuje model mapovania v centrálnej aplikácií. Je implementovaná pomocou návrhového vzoru singleton.

- *private static Mapper instance* : Jediná inštancia triedy (vzor singleton).
- *private Device[] senderDevices* : Zariadenia, ktoré sú zaregistrované v modeli s typom vysielač.
- *private Device[] receiverDevices* : Zariadenia, ktoré sú zaregistrované v modeli s typom prijímač.
- *private Mode globalMode* : Jediný globálny mód centrálnej aplikácie.
- *private Mode[] localModes* : Pole lokálnych módov centrálnej aplikácie.
- *private Mapper()* : Skrytý súkromný konštruktor (vzor singleton).
- *public Mapper getInstance()* : Vracia inštanciu triedy Mapper (vzor singleton).
- *public Boolean loadConfiguration()* : Načíta uloženú konfiguráciu centrálnej aplikácie z databázy. Vracia, či bolo načítanie úspešné.
- *public Boolean saveConiguration()* : Ukladá konfiguráciu centrálnej aplikácie do databázy. Vracia, či bolo uloženie úspešné.
- *public Device[] getDevices()* : Vracia zoznam všetkých zariadení zaregistrovaných v centrálnej aplikácii.
- *public SenderDevice[] getSenders()* : Vracia zoznam všetkých zariadení typu vysielač zaregistrovaných v centrálnej aplikácii.
- *public ReceiverDevice[] getReceivers()* : Vracia zoznam všetkých zariadení typu prijímač zaregistrovaných v centrálnej aplikácii.
- *public Device getDeviceById(String guid)* : Vracia zariadenie podľa parametra.
- *public Mode getCurrentMode()* : Vracia mód, ktorý je práve aktívny.
- *public void setCurrentMode(Mode mode)* : Nastavuje aktívny mód.
- *public Mode[] getAllModes()* : Vracia zoznam všetkých módov.
- *public void addMode(Mode mode)* : Pridá nový lokálny mód do zoznamu módov.

- *public void deleteMode(String modeName)* : Odstráni lokálny mód zo zoznamu módov.
- *public void doAction(Action action)* : Vykoná danú akciu. Tj. pošle zariadeniu, ktorému akcia patrí príkaz na jej vykonanie.
- *void senderConnected(String guid, String name, String description, String photo, Event[] events)* : Funkcia, ktorá sa volá pri pripojení zariadenia typu vysielač.
- *void receiverConnected(String guid, String name, String description, String photo, Actions[] actions)* : Funkcia, ktorá sa volá pri pripojení zariadenia typu prijímač.
- *void senderDisconnected(String name)* : Funkcia, ktorá sa volá po odpojení zariadenia typu vysielač.
- *void receiverDisconnected(String name)* : Funkcia, ktorá sa volá po odpojení zariadenia typu prijímač.
- *void eventReceived(String eventName, String deviceGuid)* : Funkcia, ktorá sa volá po prijatí udalosti od zariadenia.
- *void actionHappened(String actionName, String deviceName)* : Funkcia, ktorá sa volá po vykonaní akcie.
- *public void addSituationToMode(Event event, Action action, String modeName)* : Pridá situáciu s danou udalosťou a akciou do módu s daným názvom.
- *public void deleteSituationFromMode(Situation situation, String modeName)* : Odoberie situáciu z módu s daným názvom.
- *public void registerConnectionKinGuiInterface(ConnectionKinGuiInterface guiInterface)* : Registruje interface pre komunikáciu s používateľským rozhraním.

4.3.2 ConnectionKinGuiInterface

Interface, pomocou ktorého komunikuje Mapper s používateľským rozhraním aplikácie. Časť aplikácie, ktorá má na starosti používateľské rozhranie centrálnej aplikácie imlementuje tento listener a následne si ho zaregistruje v triede Mapper.

- *public void deviceStatusChanged(Device device)* : Funkcia, ktorá sa volá pozmene stavu zariadenia, napr. pripojení, odpojení.
- *public void eventReceived(String eventName, String deviceName)* : Funkcia, ktorá

sa volá po prijatí udalosti od zariadenia.

- *public void actionHappened(String actionName, Integer actionValue, String deviceName)* : Funkcia, ktorá sa volá po vykonaní akcie.
- *public void modeChanged(Mode actualMode)* : Funkcia, ktorá sa volá po zmene aktuálneho módu.

4.3.3 Mode

Abstraktná trieda reprezentujúca mód, v ktorom má centrálna aplikácia nastavené situácie. Mód môže byť lokálny, alebo globálny.

- *private String name* : Názov módu.
- *protected Situation[] situations* : Zoznam situácií aktívnych v tomto móde.
- *public String getName()* : Vracia názov módu.
- *void setName(String name)* : Nastavuje názov módu.
- *abstract public Boolean isGlobal()* : Vracia, či je mód globálny.
- *abstract public Situation[] getAllSituations()* : Vracia pole všetkých situácií, ktoré sú v tomto móde aktívne.
- *abstract public Situation[] getSwitchingSituations()* : Vracia pole situácií, ktoré sú v tomto móde aktívne a menia mód.

4.3.4 GlobalMode

Trieda zdedená od triedy Mode, reprezentujúca globálny mód.

4.3.5 LocalMode

Trieda zdedená od triedy Mode, reprezentujúca lokálny mód.

4.3.6 Action

Trieda, ktorá reprezentuje akciu, ktorú je schopné zariadenie vykonať.

- *private ActionType actionType* : Typ akcie: štandardná / meniaca mód.
- *private ConnKinObjectInformation* : Informácie o akcii.
- *private Device device* : Referencia na zariadenie, ktorému táto akcia patrí (ktoré si ju zaregistrovalo).

- *public ActionType getActionType()* : Vracia typ akcie.
- *void setActionType(ActionType actionType)* : Nastavuje typ akcie.
- *public ConnKinObjectInformation getObjectInformation()* : Vracia objekt s informáciami o udalosti.
- *public Device getDevice()* : Vracia objekt reprezentujúci zariadenie, ktorému udalosť patrí.
- *void setDevice(Device device)* : Nastavuje zariadenie, ktorému udalosť patrí.

4.3.7 ActionType

Výčtový typ reprezentujúci typ akcie: štandardná, meniaca mód.

- STANDARD : štandardná akcia.
- SWITCHING : akcia meniaca mód aplikácie.

4.3.8 Event

Trieda, ktorá reprezentuje udalosť, ktorú je schopné zariadenie vysielať.

- *private ConnKinObjectInformation* : Informácie o udalosti.
- *private Device device* : Referencia na zariadenie, ktorému táto udalosť patrí (ktoré si ju zaregistrovalo).
- *public ConnKinObjectInformation getObjectInformation()* : Vracia objekt s informáciami o udalosti.
- *public Device getDevice()* : Vracia objekt reprezentujúci zariadenie, ktorému udalosť patrí.
- *void setDevice(Device device)* : Nastavuje zariadenie, ktorému udalosť patrí.

4.3.9 Device

Abstraktná trieda, ktorá reprezentuje zariadenie, ako napríklad Kinect, infračervený vysielač, hifi, telefón atď.

- *private String guid* : Globálne unikátny identifikátor zariadenia.

- *private DeviceType deviceType* : Typ zariadenia.
- *private ConnKinObjectInformation* : Informácie o zariadení.
- *public String getGuid()* : Vracia unikátny identifikátor zariadenia reprezentovaného objektom tejto triedy.
- *void setGuid()* : Nastavuje unikátny identifikátor zariadenia reprezentovaného objektom tejto triedy.
- *public Boolean isSender()* : Vracia, či je toto zariadenie vysielač (sender).
- *void setDeviceType(DeviceType deviceType)* : Nastavuje typ zariadenia.
- *public Boolean isConnected()* : Vracia, či je zariadenie aktuálne pripojené.
- *public ConnKinObjectInformation getObjectInformation()*: Vracia objekt s informáciami o zariadení.

4.3.10 DeviceType

Výčtový typ reprezentujúci typ zariadenia: prijímač, vysielač.

- SENDER : typ vysielač
- RECEIVER : typ prijímač

4.3.11 SenderDevice

Trieda zdedená od triedy Device, ktorá reprezentuje zariadenie typu vysielač, napr. Kinect, telefón atď. Tieto zariadenia dokážu posielať centrálnej aplikácii rôzne udalosti.

- *private Event[] events* : Zoznam udalostí, ktoré je zariadenie schopné posielať.
- *public Event[] getEvents()* : Vracia zoznam udalostí, ktoré je zariadenie schopné posielať.
- *void setEvents(Event[] events)* : Nastavuje zoznam udalostí, ktoré je zariadenie schopné posielať.

4.3.12 ReceiverDevice

Trieda zdedená od triedy Device, ktorá reprezentuje zariadenie typu prijímač, napr. infračervený vysielač, televízor atď. Tieto zariadenia dokážu od centrálnej aplikácie prijímať rôzne akcie, ktoré následne vykonajú.

- *private Action[] actions* : Zoznam akcií, ktoré je zariadenie schopné vykonať.
- *public Action[] getActions()* : Vracia zoznam akcií, ktoré je zariadenie schopné vykonať.
- *void setActions(Action[] actions)* : Nastavuje zoznam akcií, ktoré je zariadenie schopné posielať.

4.3.13 ConnKinObjectInformation

Táto trieda spája informácie ako meno, popis a obrázok o rôznych objektoch používaných v projekte ako akcie, udalosť a zariadenie. Tieto informácie by sa v každej z týchto tried opakovali a sú preto zhrnuté v tejto triede.

- *public String getName()* : Vracia hodnotu názvu objektu ako textový reťazec.
- *public void setName(String name)* : Nastaví hodnotu názvu objektu podľa parametra.
- *public String getDescription()* : Vracia hodnotu popisu objektu ako textový reťazec.
- *public void setDescription(String description)* : Nastaví hodnotu popisu objektu podľa parametra.
- *public String getPhoto()* : Vracia hodnotu obrázku objektu ako textový reťazec.
- *public void setPhoto(String photo)* : Nastaví hodnotu obrázku objektu podľa parametra.

4.3.14 Situation

Táto trieda reprezentuje situáciu, teda dvojicu udalosť a akcia, ktoré si používateľ namapuje v centrálnej aplikácii.

- *private Event event* : Udalosť, ktorá spúšťa situáciu.
- *private Action action* : Akcia, ktorú situácia vykonáva.

- *public Event getEvent()* : Vracia udalosť, ktorá spúšťa situáciu.
- *void setEvent(Event event)* : Nastavuje udalosť, ktorá spúšťa situáciu.
- *public Action getAction()* : Vracia akciu, ktorú situácia vykonáva.
- *void setAction(Action action)* : Nastavuje akciu, ktorú situácia vykonáva.
- *public Device getSourceDevice()* : Vracia zariadenie, ktorému patrí udalosť spúšťajúca situáciu (ktoré si ju zaregistrovalo).
- *public Device getTargetDevice()* : Vracia zariadenie, ktorému patrí akcia spúšťaná situáciou (ktoré si ju zaregistrovalo).