Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Monitorace koncentrace glukózy pomocí nositelných zařízení

Plzeň 2018 Bc. Martin Úbl

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 16. května 2018

Bc. Martin Úbl

Abstract

The main goal of this paper is to analyze ways of measuring glucose levels in blood and subcutaneous tissue and methods of transferring measured levels from sensor to another device, such as mobile phone. The paper further analyzes a design of universal architecture, which would function with real device, and may also be used for simulations within one implementation.

The first part of this work is focused on diabetes mellitus, glucose homeostasis in human body, glucose level measurement and complications connected with diabetes. Then comes a chapter dedicated to sensors, data transfer and analysis. The end of part two is dedicated to modeling of glucose dynamics in human body.

The second park of this work is focused on architecture design, which complies with given requirements – is able to work either with real device and perform calculations on i.e. mobile phone or with already measured data in a simulation.

Abstrakt

Hlavním cílem této práce je analýza problematiky měření hladiny glukózy v krvi a v podkoží a možností přenosu měřeného signálu ze senzoru do dalších zařízení, například mobilního telefonu. Dále se práce zabývá analýzou a návrhem univerzální architektury, která umožní pracovat jak s reálným zařízením, tak v režimu simulace, a to v rámci jedné implementace.

V první části se práce zabývá problematikou nemoci diabetes mellitus, homeostázou glukózy v lidském těle, měřením hladiny glukózy a komplikacemi s touto nemocí spjatými. Dále následuje kapitola věnující se měřicím přístrojům, přenosu naměřených dat a možnostem zpracování. Poté je krátce rozebrána problematika modelování dynamiky glukózy v lidském těle.

Druhá část práce je věnována návrhu architektury, která vyhovuje zmíněným požadavkům – jednou implementací lze provádět jak měření reálným zařízením a výpočty např. na mobilním telefonu, tak simulace pomocí již naměřených dat na desktopovém počítači.

Poděkování

Rád bych touto cestou poděkoval Doc. Ing. Tomáši Koutnému, Ph.D. za odborné vedení diplomové práce, cenné rady a čas věnovaný při konzultacích.

Obsah

1 Úvod

Diabetes je onemocnění, kterým trpí značná část světové populace. Vyznačuje se zvýšenou hladinou koncentrace glukózy v krvi. Vzhledem k tomu, že příznaky jsou pro běžného člověka zprvu nenápadné, o něm mnozí ani nemusí vědět, dokud není příliš pozdě – neléčený diabetes může vést k vážným zdravotním komplikacím vlivem poškození orgánů, potažmo ke smrti.

Pro optimální léčbu diabetu je nutné mít dostatek informací o pacientovi a jeho těle, z nichž nejdůležitější je okamžitá hodnota glukózy v krvi, která je rozhodující pro související děje v lidském těle. Prakticky se ale glukóza měří pravidelně v krátkých intervalech v tkáňovém moku (typicky v podkožní tkáni), v krvi je měřena pouze sporadicky, obvykle dvakrát denně kvůli správné kalibraci senzoru. Měřené hodnoty z tkáňového moku jsou však jiné, než jsou ve skutečnosti v krvi. Z toho vyplývá, že je potřeba najít co nejpřesnější model, který bude co nejlépe vystihovat dynamiku glukózy v těchto dvou prostředích, aby bylo možné správně odhadovat hladinu glukózy v krvi.

Najít takový model je ale obtížný úkol a pro usnadnění vývoje je potřeba mít odpovídající programové vybavení, které umožní navržený model snadno implementovat, experimentovat s ním a porovnávat s jinými modely. Prakticky je ale nutné výsledky této práce dostat i k pacientovi s diagnostikovaným diabetem, aby bylo možné na základě těchto informací léčbu zlepšit.

Cílem této práce je vytvořit takové programové vybavení, které splňuje zmíněné požadavky – je simulační platformou pro modely dynamiky glukózy v těle a zároveň lze přenést hotový model až k samotnému pacientovi, například formou aplikace pro mobilní telefon. Výstup této práce by měl nahradit doposud používaný program *gpredict2* vyvinutý vedoucím práce.

2 Diabetes

Diabetes mellitus, česky *úplavice cukrová*, hovorově také *cukrovka*, je heterogenní skupina onemocnění, která má společný rys – zvýšenou hladinu glukózy v krvi. Hladina glukózy v krvi je přirozeně regulována lidským tělem tak, aby se pohybovala v rozmezí $3.5-5.5 mmol/l$ [12]. Běžně ale hodnota může tyto meze překročit, u zdravých lidí však pouze krátkodobě.

2.1 Distribuce glukózy v těle

Glukóza, chemickým vzorcem $C_6H_{12}O_6$, je monosacharid. Spolu s fruktózou a galaktózou tvoří trojici monosacharidů, které se přímo vstřebávají do krve při trávení potravy [33]. Fruktóza a galaktóza je metabolizována téměř výlučně v játrech a je přeměňována na glukózu. Ta pak může být přeměněna na glykogen, jakožto polysacharid, ve kterém tělo ukládá energii. Hlavním zdrojem energie je však glukóza, která je metabolizována v buňkách ve zbytku těla.

Do těla se glukóza dostává potravou a to buď volně v podobě monosacharidu (to zahrnuje i fruktózu a galaktózu, která se kaskádou reakcí v těle na glukózu v konečném důsledku také přemění), nebo vázaně v podobě disacharidů nebo polysacharidů. Během procesu trávení jsou veškeré složitější formy cukrů rozloženy na jednoduché monosacharidy, které se do krevního oběhu vstřebávají ve střevech pomocí glukózových transportérů.

O regulaci hladiny glukózy v krvi se stará primárně slinivka břišní a játra. Roli těchto orgánů při regulaci hladiny glukózy dále rozebírá kapitola 2.2. V krvi se tedy nachází glukóza v takové koncentraci, aby bylo možné zásobit veškeré buňky takovým množství energie, jaké je třeba pro jejich správnou funkci. Přenos glukózy z krve do jednotlivých buněk je proces rozdělený typicky na dvě fáze.

První fází je přenos glukózy mezi cévní soustavou a takzvaným instersticiem, které představuje mezibuněčný prostor vyplněný řídkým měkkým vazivem a tekutinou zvanou tkáňový mok[1]. Tento přenos probíhá skrze stěnu kapi-

[1] častěji *intersticiální tekutina*

lár v rámci procesu zvaném *mikrocirkulace*[13]. Ten zahrnuje obousměrný přenos, kdy koncentrace v obou prostředích mají tendenci se vyrovnávat.

Druhou fází je pak přenos glukózy skrze buněčnou stěnu do intracelulární tekutiny, tedy do nitra buněk. O tento proces se starají glukózové transportéry na povrchu buněk společně s inzulinovými receptory, které přenos přes buněčnou membránu umožňují. Jedná se tedy o proces zvaný *usnadněná difuze*[2][13].

Glykogen

Jako zásobní forma glukózy v lidském těle slouží polysacharid zvaný glykogen. Ten je vytvářen v játrech z glukózy v procesu zvaném *glykogeneze*. Následně je ukládán v játrech a v malé míře i ve svalech. V případě že je v těle nízká koncentrace glukózy v krvi, slouží jako její zdroj v opačném procesu zvaném *glykogenolýza*, kdy je naopak z glykogenu vytvářena glukóza.

2.2 Homeostáza glukózy

Dlouhodobá snaha těla udržet hladinu glukózy v krvi ve zmíněném rozsahu se nazývá *homeostáza glukózy*. O tento proces se starají dva orgány v lidském těle – slinivka břišní a játra.

V homeostáze glukózy se uplatňují zejména dva hormony – inzulin a glukagon. Inzulin umožňuje transport glukózy skrze glukózové transportéry dovnitř buněk, což má za efekt snížení koncentrace glukózy v intersticiální tekutině. Také podporuje *glykogenezi*, tedy přeměnu glukózy na zásobní glykogen. Glukagon je hormon podporující tzv. *glykogenolýzu*. To je proces, při kterém je přeměňován zásobní glykogen zpět na glukózu. Efektem je tedy zvýšení koncentrace glukózy v krvi. Inzulin i glukagon jsou produkovány ve slinivce břišní – inzulin v β-buňkách a glukagon v α-buňkách Langerhansových ostrůvků.

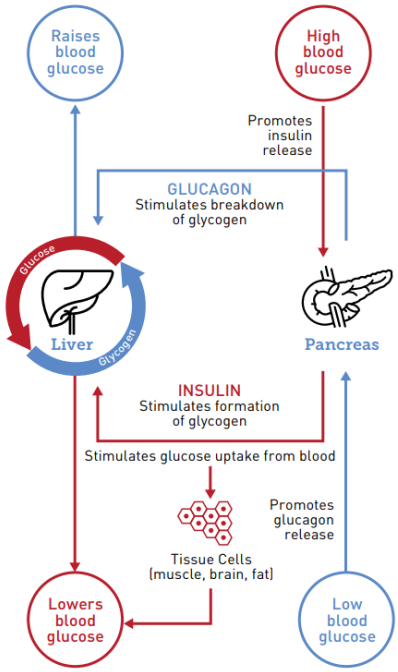
Jak bylo zmíněno v předchozí kapitole, glukóza je vstřebávána do krve stěnou střeva. Odtud se dostává přímo do krve a v tento moment se již začínají uplatňovat regulační mechanismy slinivky a jater. V případě, že je slinivkou

[2] někdy „pasivní difuze“, anglicky *facilitated diffusion*

detekována zvýšená koncentrace glukózy v krvi, je uvolňován inzulin s cílem tuto koncentraci snížit[13]. Na tento hormon také reagují játra, která podle koncentrace inzulinu v oběhu provádějí v odpovídající míře *glykogenezi*, snižující koncentraci hladiny glukózy v krvi.

Produktem glykogeneze je glykogen, který je ukládán v játrech a svalech pro případ, že by koncentrace glukózy v krvi klesla pod určitou hladinu. Pokud k takové situaci dojde, slinivka břišní uvolní hormon glukagon, který spustí *glykogenolýzu* a tím koncentraci glukózy v krvi opět zvedne.

Schematicky lze proces homeostázy glukózy vidět na obrázku 2.1.

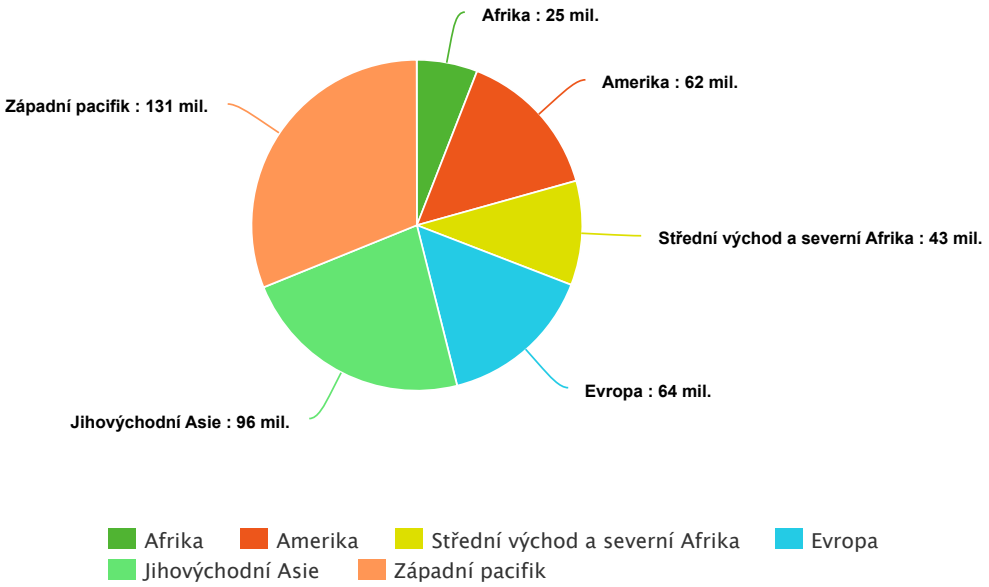


Obrázek 2.1: Znázornění role slinivky břišní a jater při homeostázi glukózy. Obrázek převzat z [16]

2.3 Diabetes

Dlouhodobě zvýšená koncentrace glukózy v krvi a poruchy homeostázy glukózy jsou společným příznakem všech onemocnění z heterogenní skupiny *diabetes mellitus*. Příčiny se však liší s každým typem tohoto onemocnění.

Dle Světové zdravotnické organizace[3] trpí diabetem na světě více než 8.5 % dospělých lidí[11]. Rozložení diabetu v různých částech světa lze vidět na obrázku 2.2, který znázorňuje absolutní podíl jednotlivých oblastí. V České republice pak Diabetická asociace ČR uvádí, že jde o 858 tisíc lidí v roce 2015, tedy vztaženo k celkovému počtu obyvatel tohoto roku jde o 8.1 % obyvatel[9].



Obrázek 2.2: Absolutní zastoupení diabetu ve světové populaci v jednotlivých regionech. Zdroj dat je [11]

Dále dle Světové zdravotnické organizace od roku 1980 vzrostl počet lidí zasažených jakoukoliv formou diabetu na dvojnásobek[11]. Tuto statistiku potvrzuje i Mezinárodní diabetická federace[4] a doplňuje navíc prognózu, že do roku 2048 se tento počet dále zdvojnásobí[16]. Tento nárůst se dá vysvětlit zejména špatnými tendencemi v životosprávě, kdy lidé konzumují čím dál tím více nezdravých jídel spojených s obezitou a například vlivem urbanizace často provozují sedavý styl života. Tyto faktory podstatně zvyšují rizika vzniku diabetu 2. typu, který bude popsán v následujících kapitolách.

Diabetes je nebezpečný zejména tím, že vysoká hladina glykemie má nežádoucí účinky, když není nijak regulována – nadbytečná glukóza se zachytává

[3]World Health Organization, WHO
[4]International Diabetes Federation, IDF

na orgány a poškozuje je. Poškození se vztahuje například na ledviny a jejich funkci filtrovat, sítnici oka, centrální nervový systém a další. Dále ačkoliv je hladina glukózy vysoká, do buněk neproniká kvůli nedostatku inzulinu nebo inzulinové rezistenci. Tato skutečnost přinutí buňky získat energii z jiných zdrojů, například rozkladem tuku. Při tomto procesu ale vznikají látky zvané ketony, které okyselují prostředí v organismu, což vede k takzvané diabetické ketacidóze.

Svými příznaky je diabetes zprvu nenápadný, jelikož v krátkodobém hledisku nejsou jeho projevy nijak výrazné. Tyto projevy zahrnují soustavnou žízeň a hlad a s tím spojené častější močení, dále zhoršené vidění, únavu a úbytek na váze. S neléčeným diabetem se mohou časem objevit další komplikace, jako je snížená citlivost dolních končetin s výskytem gangrén (symptom známý jako „diabetická noha“), ovlivnění metabolických drah vlivem ketacidózy, poškození sítnice, které může vést až ke slepotě, a další projevy, které vznikají vlivem zmíněných změn v organismu. Tělo je také celkově náchylnější k infekcím.

Původ diabetu neni doposud přesně znám. Je ale známo, že předpoklady ke vzniku diabetu jsou genetické a k jeho rozvoji přispívá právě nezdravý životní styl.

2.3.1 Chronické komplikace

Mezi nejčastější chronické komplikace se řadí:

- poškození periferních nervů (*neuropatie*)
- poškození jater (*nefropatie*)
- poškození oční sítnice (*retinopatie*)
- poškození dolních končetin (diabetická noha)
- ketacidóza

Z těchto komplikací je nejtypičtější poškození periferních nervů, jelikož postihuje téměř všechny diabetiky po relativně krátké době trvání nemoci. Začíná nejčastěji obyčejným brněním a mravenčením, později přechází k poruše citlivosti a postupnému celkovému snížení citlivosti postižené části periferních nervů.

2.3.2 Akutní komplikace

Diabetes mellitus může ve svém trvání způsobovat dva rizikové stavy, které jsou považovány za akutní komplikaci. Jedná se o stav, kdy je koncentrace glukózy větší, než horní hranice, nebo stav, kdy je naopak menší, než spodní hranice. Takový stav se nazývá hyperglykemie, respektive hypoglykemie. To přináší nutnost měření koncentrace glukózy, aby bylo možné odpovídajícím způsobem dávkovat příslušné léčivo.

Hypoglykemie

Jako hypoglykemie se označuje stav, kdy je hodnota koncentrace glukózy v krvi pod hranicí 3.3 $mmol/l$. Při takových hodnotách nejsou buňky zásobovány dostatkem energie, což vyúst'uje v celkovou fyzickou i psychickou slabost, bolest hlavy, poruchu jemné motoriky a v horších případech pak vede ke stavu, který se označuje jako *hypoglykemické kóma*, tedy stav bezvědomí zapříčiněný kriticky nízkou koncentrací glukózy v krvi. Pokud není tento stav regulován např. dávkou glukagonu nebo jinou odpovídající medikací, a tělo se s ním není schopno vyrovnat samo, může dojít k trvalému poškození orgánů.

Příčinou může být velká fyzická zátěž, stres, alkohol, nebo například i zkonzumování velkého množství jídla, kdy tělo vyprodukuje příliš mnoho inzulinu naráz. Kromě těchto příčin může být hypoglykemie indukovaná i některými léky. Také může tento stav vzniknout při aplikaci větší dávky inzulinu, než je skutečně nutná.

Hyperglykemie

Jako hyperglykemie se označuje naopak stav, kdy je hodnota koncentrace glukózy v krvi nad hranicí 11.1 $mmol/l$. Při takové koncentraci již může docházet ke zmíněným komplikacím a dlouhodobě i k poškození orgánů.

Okamžitou léčbou hyperglykemie je aplikace inzulinu, a to bud' inzulinovým perem ručně nebo například inzulinovou pumpou, která je napojena na měřící přístroje a dávkuje inzulin automaticky.

Vzhledem k tomu, že hladina cukru v těle diabetika má vždy tendenci být

vyšší, může být příčinou trvající hyperglykemie například opomenutí aplikace dávky inzulinu, případně porucha inzulinové pumpy nebo jejího propojení se senzorem nebo lidským tělem.

2.3.3 Typy diabetu

Podle příčiny těchto poruch je zavedeno dělení na několik hlavních typů. Dochází buď k absolutnímu nebo relativnímu nedostatku inzulinu, jakožto hormonu regulujícím pronikání glukózy do buněk, respektive redukujícím koncentraci glukózy v krvi.

Základní dělení hovoří o typu 1, typu 2, gestačním diabetu a skupině zvané sekundární diabetes. Nová studie však hovoří o jednoznačně identifikovatelných pěti typech diabetu v závislosti na míře rezistence na inzulin[4].

Typ 1

Tento typ diabetu je autoimunitním onemocněním. Imunitní systém postiženého organismu, který se za normálních okolností stará pouze o izolaci nebo likvidaci cizích a potenciálně nebezpečných látek, napadá i buňky, které by napadat neměl – konkrétně jsou likvidovány β-buňky Langerhansových ostrůvků[13] produkujících mimo jiné i inzulin, který je zodpovědný za snižování koncentrace glukózy v krvi. Jde tedy ve výsledku o absolutní nedostatek inzulinu.

Diabetes typu 1 je obvykle diagnostikován v dětství nebo při dospívání. Není ale vyloučena jeho pozdní forma, která se u člověka vyskytuje po třicátém roku života.

Vzhledem k tomu, že tělo diabetika s tímto typem nedokáže samo produkovat inzulin, je doživotně odkázán na léčbu inzulinem dodávaným v injekcích (inzulinových perech) nebo inzulinovou pumpou. Současně s tím je potřeba soustavně monitorovat koncentraci glukózy, aby bylo možné ji včas a správně regulovat.

Typ 2

Typ 2 se vyznačuje inzulinovou rezistencí, tedy sníženou reaktivitou inzulinových receptorů na povrchu buněk, které dovolují glukóze prostupovat buněčnou membránou. Inzulinu tedy je produkován dostatek, ale nedostavuje se příslušná odpověď zbytku organismu, která by koncentraci glukózy vyrovnala. Slinivka, konkrétně β-buňky Langerhansových ostrůvků jsou tedy v nadprodukci. Časem tato skutečnost přejde až k vyčerpání a selhání β-buněk, a tedy ke snížení produkce vlastního inzulinu[13].

Tento typ diabetu je diagnostikován obvykle ve věku nad 30 let a je spojován s nezdravým životním stylem. Výskyt u dětí a mladistvých však není vyloučen. Většina pacientů s diagnostikovaným diabetem má diagnostikován právě tento typ[16].

Tělo diabetika s typem 2 může být stále schopno produkovat inzulin, takže další dodávání inzulinu není univerzálním řešením. Místo toho je pacientovi doporučena striktní dieta a zpravidla je mu předepsán některý ze skupiny léků zvaných *antidiabetika*. Tyto léky buď zvyšují produkci vlastního inzulinu, zvyšují citlivost buněk na inzulin nebo například blokují vstřebávání monosacharidů ze střev. Inzulin je ale pochopitelně třeba podávat v případě, že už byla ve větší míře poškozena tkáň, která ho produkuje.

Ostatní typy

Mezi další typy se řadí například *gestační diabetes*, který vypukne a přetrvává pouze v těhotenství. Koncentrace glukózy v krvi pacientky je zvýšená, jelikož je zvýšena i rezistence na inzulin. Tento typ diabetu je rizikový jednak pro zdraví pacientky z podobných důvodů jako u diabetu typu 2, a jednak pro vývoji plodu.

Další typy diabetu se souhrnně označují jako *sekundární diabetes*. Do této kategorie spadají všechny druhy diabetu, které vznikají vlivem jiné nemoci nebo například medikací, která je podávána na její léčbu.

3 Modelování dynamiky glukózy

Pro tělo je důležitá koncentrace glukózy v krvi, ale ta v běžných podmínkách není měřena kontinuálně – pouze sporadicky v dlouhých časových intervalech. Jediný běžně dostupný kontinuálně měřený signál je právě ten z intersticiální tekutiny. Existují však modely, které jsou schopné na základě obou těchto signálů stanovit koncentraci v krvi kontinuálně. Některé z nich jsou schopné i předpovídat.

Běžně používaným modelem je například model Steil-Rebrinové. Dále je vedoucím práce vyvíjen model difúzní. Existují i další modely, ale tyto dva staví na zmíněných běžně dostupných sinálech.

3.1 Model Steil-Rebrinové

Model Steil-Rebrinové byl prvním modelem, který po nástupu technologií pro kontinuální měření vykazoval uspokojivě nízkou chybovost. Dynamiku glukózy popisuje vztahem[31]:

$$\frac{di(t)}{dt} = -q_2 i(t) + q_1 b(t), \tag{3.1}$$

kde $i(t)$ je koncentrací glukózy v intersticiální tekutině, $b(t)$ koncentrací glukózy v krvi a parametry q_1 a q_2 popisují zpoždění při difuzních jevech. Vedoucí práce pro další výzkum používá tento model s modifikací dle Del Favera[10], integrující model chyby senzoru, který počítá se zkreslením při kalibraci. Výsledný model s vyjádřením $b(t)$ tedy popisuje rovnice:

$$b(t) = \frac{\tau}{\alpha} \cdot \frac{di(t)}{dt} + \frac{1}{\alpha} \cdot i(t) - \frac{\beta + \tau \cdot \gamma}{\alpha} - \frac{\gamma}{\alpha} \cdot \Delta t(t) \tag{3.2}$$

Parametry α, β a γ jsou tzv. kalibrační parametry (mění se při každé kalibraci), τ je koeficientem „vyrovnávání“ hladiny koncentrace v krvi a v intersticiální tekutině a $\Delta t(t)$ je funkcí, která má v daném čase hodnotu časového

rozdílu od poslední kalibrace (lze tedy uvažovat, že $\Delta t(t) = t - t_{cal}$, kde t_{cal} je čas poslední kalibrace).

3.2 Difúzní model

Model Steil-Rebrinové má však nedostatky, které popisuje studie [20]. Kvůli těmto nedostatkům byl vedoucím práce navržen difúzní model dynamiky glukózy, který obsahuje více parametrů a zahrnuje více jevů v lidském těle. Základem modelu jsou tyto vztahy[20]:

$$\begin{aligned} \varphi(t) &= t + \Delta t + k \cdot \frac{(i(t) - i(t-h))}{h} \\ p \cdot b(t) + cg \cdot b(t) \cdot (b(t) - i(t)) + c &= i(\varphi(t)) \end{aligned} \tag{3.3}$$

Zde $b(t)$ označuje okamžitou koncentraci glukózy v krvi, $i(t)$ okamžitou koncentraci glukózy v intersticiální tekutině, parametr p popisuje zisk glukózy z krve vlivem mezibuněčných jevů, parametr cg popisuje difúzní prostředí v podobě plochy membrány a permeability prostředí, parametr c je průměrná reziduální koncentrace glukózy, parametr k popisuje změnu koncentračního gradientu a parametr h interval aplikace parametru k.

Lze si všimnout, že model zachycuje vztah okamžité koncentrace glukózy v krvi a v intersticiální tekutině s koncentrací v intersticiální tekutině v budoucnu (o Δt dále).

Pro výpočet koncentrace glukózy v krvi lze z výše uvedeného vzorce vyjádřit $b(t)$, to ovšem vede na kvadratickou rovnici:

$$cg \cdot b^2(t) - (cg \cdot i(t) - p) \cdot b(t) + c - i(\varphi(t)) = 0 \tag{3.4}$$

Pokud parametr cg není roven nule, rovnice je tedy kvadratická a má právě dvě reálná řešení. Uvažujeme ale pouze řešení kladné[20]:

$$b(t) = \frac{cg \cdot i(t) - p + \sqrt{(cg \cdot i(t) - p)^2 - 4 \cdot cg \cdot (c - i(\varphi(t)))}}{2 \cdot cg} \tag{3.5}$$

Pokud je *cg* rovno nule, je rovnice degradována na lineární:

$$b(t) = \frac{c - i(\varphi(t))}{p} \tag{3.6}$$

Model je již navržen jako predikční, proto je možné zachovat původní tvar rovnice 3.3 pro předpověď koncentrace glukózy v intersticiální tekutině. Vyvstává ale problém odlišný – vzhledem k faktu, že model předpovídá hodnoty, je nutné analyzovat, jaký časový úsek do budoucnosti bude mít dostatečnou přesnost.

3.3 Další modely

Mimo zmíněné modely existují i další, které ovšem často staví na údajích v běžných podmínkách nedostupných (například údaje o produkci inzulinu). Z těchto modelů jde například o Hovorkův model, který modeluje vztah koncentrace glukózy v intersticiální tekutině a v krvi pomocí dynamiky inzulinové absorpce v daných prostředích[14]. Tento model je ale příliš komplexní – uplatnění nachází v simulaci procesů v lidském těle.

Dále existují modely, které se neosvědčily na širší škále množin dat, jako jsou například neuronové sítě a statistické modely. Modelování pomocí neuronových sítí ovšem nevykazovalo potřebnou přesnost při předpovědi vývoje koncentrace glukózy v intersticiální tekutině[27].

3.4 Určení parametrů modelu

Parametry modelu není možné získat měřením – je nutné najít jiný způsob, jak je stanovit. Tato část vychází z předchozích zkušeností vedoucího práce, který používá zejména dva přístupy: nelineární optimalizací algoritmem NEWUOA a metadiferenciální evoluci.

Problém hledání parametrů modelu je optimalizačním problémem. Hledání parametrů tedy probíhá za minimalizace funkce zdatnosti[1], která vyjadřuje míru kvality řešení (vektoru parametrů modelu). Výpočet této funkce je

[1] z angl. *fitness function*

přímo převeden na výpočet metriky pro dané řešení nad množinou naměřených dat. Metriky použité pro výpočet jsou uvedeny v kapitole 3.5.

3.4.1 NEWUOA

Algoritmus s názvem *NEWUOA* je optimalizačním algoritmem, který nemá horní a dolní mez. Řadí se do skupiny nelineárních algoritmů, jelikož prostor dělí na oblasti zvané „trust-region“, v rámci kterých danou funkci zdatnosti, zde označovanou jako *objective function*, interpoluje kvadratickými modely[28].

Ke své práci nepotřebuje derivaci funkce zdatnosti. Algoritmus pracuje v iteracích, kdy v každé stanoví kvadratický model pro dílčí oblasti a následně v nich minimalizuje funkci zdatnosti.

Vedoucí práce používá implementaci algoritmu v rámci knihovny NLopt[2] vytvořené na MIT[3]. Tato knihovna podporuje i další algoritmy pro nelineární optimalizaci – zejména jde pak o algoritmus BOBYQA, který má potenciál algoritmus NEWUOA pro určité problémy nahradit. Algoritmus BOBYQA je však navržen pro řešení problémů s omezením domény.

3.4.2 Metadiferenciální evoluce

Dalším kandidátem pro řešení optimalizačních problémů je algoritmus implementující evoluční strategie. Ve studii [21] byl vedoucím práce popsán algoritmus metadiferenciální evoluce v aplikaci na modely dynamiky glukózy.

Jedná se o algoritmus, který pracuje v iteracích, zde označovaných jako *generace*. Používá množinu možných řešení velikosti N, zvanou *populace*, kdy každý člen populace je vektorem parametrů daného modelu. Každá generace je dále dělena na dvě fáze: křížení a mutace. Křížením se rozumí předání části „genetické informace“ ve formě parametrů modelu ve směru od jedince s lepší hodnotou funkce zdatnosti k jedinci s menší hodnotou funkce zdatnosti. Toto křížení může probíhat dle několika různých kritérií výběru – křížení s nejlepším jedincem, s náhodným z P nejlepších jedinců ($P < N$), s náhodným z celé populace, s několika jedinci v generaci, apod. Dále je možné stanovit

[2] https://nlopt.readthedocs.io
[3] Massachusettský technologický institut

strategii křížení, kdy jedinec přebírá v různé míře a podle různého klíče parametry jedince, se kterým je křížen. To je zpravidla přičtení jeho vektoru parametrů s pevnou nebo proměnnou vahou. Mutace pak probíhá generováním nových parametrů.

Metadiferenciální evoluce oproti diferenciální evoluci navíc přenáší parametry samotné evoluce na stranu jedinců v populaci – každý jedinec s sebou nese svou strategii křížení, váhu pro křížení s jiným jedincem a pravděpodobnost, že bude nahrazen svým potomkem vzniklým křížením.

Na konci generace je přepočtena funkce zdatnosti všech jedinců v populaci a celý algoritmus se opakuje. Zastavovací podmínkou může být buď počet generací, nebo „dostatečně dobrá“ hodnota zdatností funkce. Tato hodnota by ale musela být stanovena obecně pro všechny dostupné metriky, a navíc nelze předpokládat jednotný práh pro všechny množiny dat. Je proto použito omezení na počet generací.

Po dokončení algoritmu je vybrán jedinec z poslední generace, který má nejlepší hodnotu funkce zdatnosti. Jeho vektor parametrů je použit jako nejlepší nalezené řešení parametrů daného modelu.

3.5 Metriky

Pro stanovení kvality modelu a vypočtených parametrů je nutné mít zobrazení, které číselně popisuje vzdálenost vypočtených dat od naměřených (chybu). Takovým zobrazením je metrika. Formálně je metrika definovaná jako

$$\rho : \mathcal{M} \times \mathcal{M} \to \mathbb{R} \tag{3.7}$$

pro danou množinu $\mathcal{M}$ a musí splňovat pro všechny prvky $x, y, z \in \mathcal{M}$ tyto čtyři axiomy:

1. nezápornost – $\rho(x, y) \geq 0$
2. totožnost – $\rho(x, y) = 0 \Leftrightarrow x = y$
3. symetrie – $\rho(x, y) = \rho(y, x)$

4. trojúhelníková nerovnost – $\rho(x, z) \leq \rho(x, y) + \rho(y, z)$

Pro měřená data je pak metrika definována nad množinou rozdílů naměřených a vypočtených dat. Také je možné tyto diference počítat jako absolutní nebo relativní. Pro potřeby kapitoly je množina naměřených hodnot značena jako X, množina vypočtených hodnot jako Y, střední hodnota rozdílu jako D_{mean} a mohutnost množin X a Y jako n.

3.5.1 Standardní statistické metriky

Mezi základní metriky i pro tyto potřeby patří například maximální (3.8) a průměrná hodnota (3.9) a standardní odchylka (3.13). Zde uvažujeme standardní odchylku s Besselovou korekcí.

$$\rho(X, Y) = max_{i=1}^{n}(|X_i - Y_i|) \tag{3.8}$$

$$\rho(X, Y) = \frac{\sum_{i=1}^{n}(|X_i - Y_i|)}{n} \tag{3.9}$$

$$\rho(X, Y) = \sqrt{\frac{\sum_{i=1}^{n}(|X_i - Y_i| - D_{mean})^2}{n - 1}} \tag{3.10}$$

Také je možné zavést kombinaci těchto metrik – vhodné to může být zejména u průměrné chyby a její standardní odchylky.

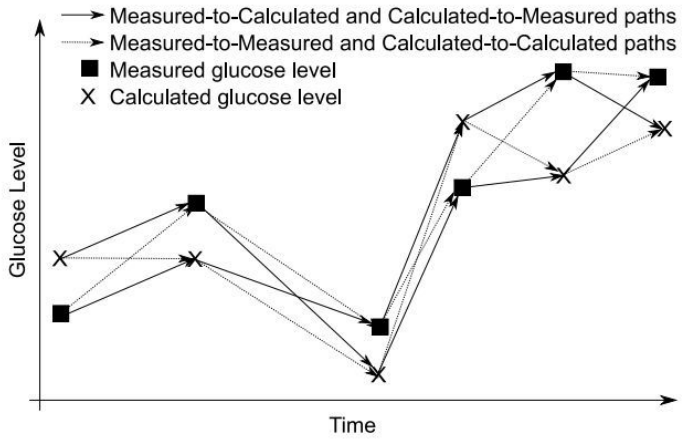
3.5.2 Crosswalk

Nedostatkem standardních metrik je zejména fakt, že nezohledňují více, než jeden rozměr. Nezáleží tedy na tom, zda byly naměřené hodnoty od sebe vzdálené pět minut nebo jeden den – metrika mezi tím nedělá rozdíl. Metriky též neuvažují pořadí hodnot (resp. diferencí), a tedy je relativně velká pravděpodobnost, že dvě různá řešení mají stejnou hodnotu metriky, aniž by měla stejnou reálnou kvalitu.

Z tohoto důvodu byla navržena metrika zvaná *crosswalk*[19]. Mějme množinu naměřených hodnot M, kde každý prvek je uspořádanou dvojicí času a

naměřené hodnoty $[t, y_m]$. Dále mějme množinu vypočtených hodnot C, kde každý prvek je uspořádanou dvojicí času a vypočtené hodnoty $[t, y_c]$. Předpokládejme, že máme naměřené a vypočtené časy identické, reálně tomu tak být nemusí, ale zavádí se vhodná aproximace nebo interpolace, aby tohoto stavu bylo dosaženo.

Pro výpočet metriky uvažujme v první iteraci spojnici první naměřené hodnoty, druhé vypočtené, třetí naměřené, a tak dále, až je dosaženo poslední hodnoty. Ve druhé iteraci vytvořme spojnici dle stejného klíče, pouze prvním bodem bude první vypočtená hodnota. Volitelně lze spojit i po sobě jdoucí naměřené hodnoty do jedné lomené čáry a všechny po sobě jdoucí vypočtené hodnoty do druhé. Vznikne tak útvar podobný tomu na obrázku 3.1.



Obrázek 3.1: Znázornění kroků výpočtu metriky crosswalk. Obrázek převzat z [19]

Výsledná metrika je součtem délek obou lomených čar vydělená počtem naměřených hodnot:

$$d_t = (t_i - t_{i-1}) \tag{3.11}$$

$$\rho(M, C) = \frac{\sum_{i=1}^{n}(\sqrt{d_t^2 + (y_{c_i} - y_{m_{i-1}})^2} + (\sqrt{d_t^2 + y_{c_{i-1}} - y_{m_i})^2})}{n} \tag{3.12}$$

Nutno dodat, že tato metrika neeliminuje možnost, že dvě odlišná řešení s výrazně odlišnou kvalitou budou mít stejnou hodnotu metriky, pouze tuto pravděpodobnost podstatně snižuje.

3.5.3 Další metriky

Mezi další významné metriky patří například Akaikeho informační kritérium (označováno zkratkou AIC), které je přímo určeno pro porovnání kvality modelů navzájem:

$$AIC(X, Y) = 2k - 2ln(\hat{\rho}(X, Y)), \tag{3.13}$$

kde $\hat{\rho}(X, Y)$ je dle definice relevantní měřítko kvality modelu, tedy například některá z uvedených metrik.

Také je možné použít například plochu mezi křivkami (resp. numerickou aproximaci například lichoběžníkovou metodou), součet čtverců rozdílů hodnot, a další metriky.

3.6 Shrnutí

V této kapitole byly krátce popsány modely dynamiky glukózy, algoritmy pro hledání jejich parametrů a metriky používané pro ohodnocení kvality řešení. Z pohledu práce jsou důležité všechny vypsané části, jelikož jedním z požadavků je právě výpočet koncentrace glukózy užitím modelu.

Práce dále uvažuje model Steil-Rebrinové a model difúzní. Z algoritmů hledání řešení uvažuje oba popsané – NEWUOA a metadiferenciální evoluci. Z popsaných metrik jsou použity všechny.

4 Přenos navzorkovaného signálu koncentrace glukózy

Koncentrace glukózy se v lidském těle v různých prostředích liší – lze se zajímat o koncentraci v kapilární krvi, v tepenné a žilní krvi, v krevní plazmě a v intersticiální tekutině.

Prakticky je prováděno měření koncentrace v žilní krvi a v intersticiální tekutině. Mezi koncentrací v krvi z různých zdrojových prostředí je sice mírný rozdíl[7], ale to pro tuto práci není relevantní. V této práci bude nadále zmiňována koncentrace glukózy v krvi vždy jako hodnota zjištěná ze vzorku získaného z píchnutí do prstu, tedy v podstatě z kapilární krve.

Hodnotu koncentrace v krvi je důležité znát jednak kvůli diagnostice diabetu, mnohem větší význam však má měření koncentrace za účelem regulace pomocí dávek inzulinu. V případě pacienta, který je odkázán na léčbu inzulinem, je potřeba měřit pravidelně hladinu koncentrace v krvi, aby bylo možné stanovit velikost *bolusu*[1]. Toto měření je však pro pacienta nepříjemné – je nutné píchnutím do prstu extrahovat drobné množství krve a přenést ho do měřicího přístroje. Píchnutí do prstu je jednak bolestivé, a jednak se nedá provádět příliš často. Podrobnosti o principu měřicích přístrojů pro měření koncentrace glukózy v krvi jsou obsaženy v kapitole 4.1.1.

Jako alternativní, ale v současné době velmi rozšířený způsob měření koncentrace glukózy, je kontinuální měření v intersticiální tekutině. Tato metoda vyžaduje na začátku měření zavedení senzoru do podkožní tkáně. Senzor má často podobu malé jehly připojené do zařízení, které se stará o přenos dat do jiných zařízení. Tím je buď přístroj na sběr dat, inzulinová pumpa nebo oboje. Tento způsob měření se označuje zkratkou CGMS[2]. Podrobnosti o principu měřicích přístrojů pro měření koncentrace glukózy v intersticiální tekutině jsou obsaženy v kapitole 4.1.2.

Hodnota koncentrace glukózy v intersticiální tekutině se ale od hladiny v krvi liší, přičemž hodnota koncentrace v krvi je pro regulaci inzulinu rozhodující – řídí se jí orgány lidského těla. Vzhledem k tomu, že přenos glukózy mezi krví

[1] z latiny: sousto; označení jednorázové dávky léku ke vpravení do žíly

[2] Continuous Glucose Monitoring System – systém pro kontinuální monitoraci glukózy

a intersticiální tekutinou probíhá skrze stěnu kapilár, a to v obou směrech, hladiny mají tendenci se vyrovnávat, ale rozhodně nelze počítat s tím, že jsou vždy stejné. Je proto potřeba transformovat toto měření modelem. Více o modelech dynamiky glukózy obsahuje kapitola 3.

Kromě výše zmíněných způsobů měření, které jsou jinak označovány jako *selfmonitoring* metody, existují například i laboratorní metody. Ty měří koncentraci v krevní plazmě nebo přímo v žilní krvi. Také je možné v laboratorních podmínkách provádět měření pomocí radioaktivního markeru.

Dále je nutné zdůraznit, že v případě měření koncentrace v krvi jde o měření sporadické, a to typicky nejvýše do 10 měření denně. V případě měření koncentrace v podkoží, respektive v intersticiální tekutině, jde o měření kontinuální. To v praxi znamená měření s periodou v jednotkách minut bez nutnosti interakce pacienta.

Měřená hodnota je obvykle reprezentována v jedné z těchto jednotek:

- mg/dl – miligramy na decilitr, jednotka používaná ve Spojených státech amerických
- $mmol/l$ – milimoly na litr, jednotka používaná ve zbytku světa

Pro zmíněné jednotky platí:

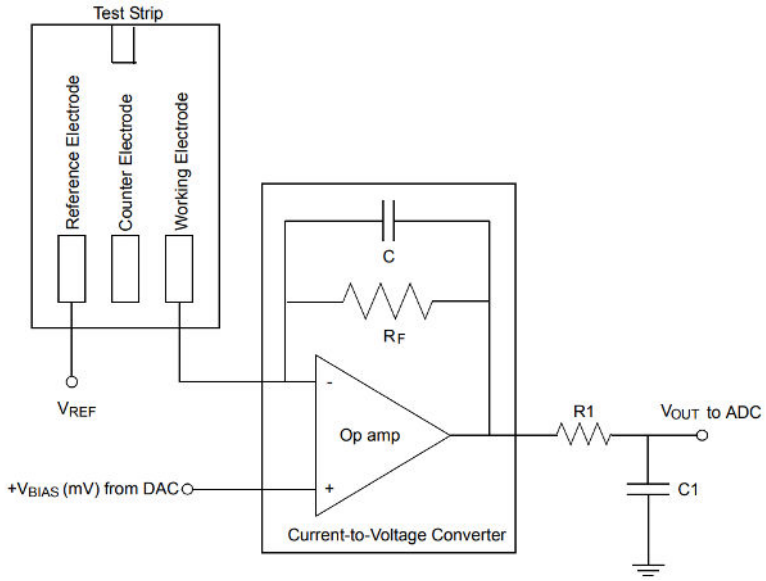
$$C[mg/dl] = C[mmol/l] * 18.018 \tag{4.1}$$

4.1 Měřicí zařízení

Pro potřeby této práce je nutné uvést pouze to, že měřicí přístroje pracují na principech buď elektrochemických nebo optických, v každém případě však poskytují do řídicí jednotky přístroje hodnotu převedenou na číslo (typicky A/D převodníkem[3])[6]. Schematicky je možné tuto část, například pro běžný glukometr pro měření hladiny cukru v krvi, vidět na obrázku 4.1.

Interpretace získaných hodnot je pak různá, ale vždy je pro přepočet na výslednou koncentraci použit model. Nejčastěji se pak jedná o lineární regresi, pro niž byly koeficienty stanoveny metodou nejmenších čtverců, jak uvádí

[3]analogově/digitální převodník, součástka pro převod spojité veličiny na diskrétní



Obrázek 4.1: Schematické znázornění měřicí části přístroje, která převádí proud vzniklý na elektrodách na napětí, které lze převodníkem transformovat na číslo. Zde v aplikaci glukometru pro měření koncentrace glukózy v krvi. Obrázek převzat z [8]

například [8]. Dle tohoto modelu jsou pak dopočítány hodnoty koncentrace glukózy z měřeného napětí. Výsledná rovnice pro přepočet převedené reprezentace měřené veličiny na koncentraci glukózy může vypadat například takto:

$$C = k \cdot U + q \tag{4.2}$$

Zde je C zjišťovaná koncentrace a U hodnota napětí (výstup převodníku). Z toho vyplývá, že senzor je potřeba zkalibrovat, aby bylo možné stanovit parametry modelu.

4.1.1 Sporadické měření - krev

Měření koncentrace glukózy v krvi vyžaduje, aby senzor byl v přímém kontaktu s krví. To implikuje nutnost extrahovat drobné množství krve, nejčastěji píchnutím do bříška prstu. Krev je pak nutné přenést na testovací proužek, který je vložen do měřicího přístroje. Každý typ testovacích proužků však může mít jiné vlastnosti, a tak není možné pro vlastní měření použít

jakýkoliv typ – pouze ten, se kterým byl přístroj kalibrován.

Samotné měření pak probíhá tak, že je po vložení vzorku vyčkána určitá krátká doba (započetí reakce), která je závislá na druhu testovacího proužku. Obvykle se tato doba pohybuje kolem 1.5 vteřiny[8, 36]. Poté řídicí jednotka spustí vzorkování hodnot z převodníku. Ve chvíli, kdy je navzorkováno dostatečné množství hodnot (například 2048, jak uvádí [8]), je vybrána reprezentativní hodnota a ta vstupuje do rovnice 4.2.

Reprezentativní hodnotou může být v jednoduchém případě průměr ze všech hodnot. Zpravidla je možné aplikovat libovolný filtr, který chyby odstíní, ale u glukometrů pro měření koncentrace glukózy v krvi nemá pokročilejší filtrování větší smysl – zašumění dat je malé a počáteční hodnoty, které prokazatelně chybovost vykazují[8], jsou vynechány vyčkáním zmíněné doby.

4.1.2 Kontinuální měření - intersticiální tekutina

Měření koncentrace glukózy v krvi je pro pacienta často nepříjemné. Jak bylo zmíněno v kapitole 2.1, hodnota koncentrace glukózy v intersticiální tekutině má tendenci se vyrovnávat s hladinou v krvi, a tak je možné měřit koncentraci zde.

Pro kontinuální měření jsou známy metody invazivní, minimálně invazivní a neinvazivní. Pro běžné „domácí“ měření je v současné době nejrozšířenější metoda invazivního měření v podkožní tkáni (kde se nachází intersticiální tekutina)[6, 13]. Tato metoda vyžaduje zavedení jehly senzoru do podkoží.

Jiné metody měření zde nebudou rozebírány, jelikož nejsou relevantní k této práci. Podstatné z pohledu této práce však je, že z technického hlediska všechny metody měří elektrický proud, který je následně přepočten na koncentraci glukózy. Tento signál se pak často v terminologii dostupných senzorů označuje jako *interstitiary signal*, příp. častěji zkratkou *ISIG*.

V případě invazivních metod je zpravidla nutné po zavedení senzoru vyčkat určitou dobu stanovenou výrobcem, než je možné provádět měření. První perioda po zavedení senzoru slouží pro „zaplavení“ elektrody tkáňovým mokem a je dlouhá nejvýše několik jednotek minut. Následuje perioda delší, během které je senzor nečinný. Takto zavedený senzor totiž vyvolává reakce lidského těla typické pro zranění a poškození tkáně – jedná se v podstatě o cizí těleso, které narušilo pokožku[6]. Během této doby se mohou vlastnosti okol-

ního prostředí senzoru měnit, a tím by mohla měření ztrácet na přesnosti. Tato perioda trvá zpravidla jednotky hodin.

Kalibrace senzoru

Uvažujme nyní senzor, který je správně zaveden a již poskytuje hodnoty ISIG, které lze považovat za správné. Nyní je nutné provést kalibraci, aby bylo možné správně převádět měřený signál na koncentraci glukózy. Obdobně jako u měření krve je nutné mít referenční hodnoty, které senzor použije pro stanovení parametrů modelu. U kalibrace glukometru však tyto hodnoty pocházely z měření roztoků o známé koncentraci glukózy. Zde takový druh kalibrace použít nemůžeme – senzor bychom museli kalibrovat mimo tělo v jiných podmínkách, a to u drtivé většiny metod není možné. Také bychom kalibrovali v prostředí s jinými parametry, než jsou skutečné pracovní, a tedy by výsledná kalibrace byla špatná.

Jako referenční hodnota je proto použita koncentrace glukózy v krvi zjištěná glukometrem, nebo jiným, již zkalibrovaným přístrojem. Vzhledem k difuzním jevům popsaných v kapitole 2.1 se koncentrace glukózy v krvi a v intersticiální tekutině má tendenci vyrovnávat. To implicitně umožňuje použít koncentraci glukózy v krvi ke kalibraci senzoru, který měří koncentraci v intersticiální tekutině, jelikož rozdíl těchto koncentrací by neměl být nijak vysoký.

Hlavní problém této kalibrace ale je zanedbání skutečnosti, že koncentrace glukózy v krvi a v intersticiální tekutině se zpravidla liší. Je sice získán dostatečně přesný přepočet, ovšem pro jakékoliv další výpočty je nutné tuto skutečnost vzít v potaz. Existují metody, které se snaží minimalizovat tuto nepřesnost, jako například uvažování zpoždění mezi těmito hladinami nebo zavedení tzv. Kalmanova filtru[3]. Pro potřeby této práce však není tyto metody nutné dále rozvíjet.

Senzor je často použitelný pouze v řádu jednotek dnů, a to jednak kvůli degradaci samotného senzoru, a jednak kvůli tomu, že imunitní systém těla má tendenci se „zbavovat“ cizího tělesa, kterým takto invazivní senzor je[18].

Filtrace dat kontinuálního měření

Pro snížení nepřesností měření, kalibrace a minimalizaci vlivu šumu je pro kontinuální měření dále zaváděno filtrování hodnot na výstupu. Výstupem senzoru je sice obvykle jedna hodnota za např. 5 minut, ve skutečnosti však senzor měří několikrát během této doby a až výstupní filtr generuje reprezentativní hodnotu z uplynulé měřicí periody[3]. Počet dílčích měření a jejich perioda v rámci tohoto času, stejně jako perioda celá je specifická pro každý model senzoru.

Lze aplikovat různé druhy filtrů – například filtr mediánový, filtr s konečnou impulzní odezvou (klouzavý průmer), filtr s nekonečnou impulzní odezvou a další. Podstatnou informací však je, že poskytnutá hodnota se nemusí rovnat žádné reálné naměřené hodnotě, jde pouze o vybranou reprezentativní hodnotu, která je nejlepším odhadem.

4.2 Přenos dat

Ekosystém zařízení, která se podílejí na měření glukózy nekončí u glukometrů a senzorů zavedených v podkoží. Zejména u kontinuálního měření je nutné oddělit senzor od řídicí jednotky, která se stará o vyhodnocení a případně o dávkování inzulinu.

Systém pro kontinuální měření bývá nejčastěji dělen na tři části – senzor, vysílač a řídicí jednotka. Funkce senzoru byla popsána v předchozích kapitolách. Ten je s vysílačem propojen vždy přímo užitím konektoru (specifický pro každého výrobce). V této kapitole bude věnována pozornost propojení vysílače s dalším zařízením, ať jde o výrobcem dodávaný přístroj, mobilní telefon nebo jiné zařízení.

Všechny moderní přístroje pro kontinuální měření obsahují bezdrátové technologie. V minulosti sice přístroje obsahovaly i technologie drátové, ale od nich bylo z důvodů komfortu upuštěno.

4.2.1 Bezdrátové spoje

Mnohem větší komfort pro pacienta přináší spoje bezdrátové. Ty však podléhají mnohem přísnějším požadavkům, které musí být splněny, pokud má být zařízení schváleno pro oficiální používání[30].

Hlavními požadavky na bezdrátové spoje v medicínských aplikacích jsou[25]:

- interoperabilita – schopnost vysílače pracovat se širokou škálou zařízení i od jiných výrobců
- nízká spotřeba – senzory a vysílače jsou obvykle provozovány na baterii a pro delší výdrž je nutné optimalizovat i protokolový zásobník
- přizpůsobení protokolového zásobníku medicínským aplikacím
- kompatibilita – zařízení nesmí narušovat činnost jiných zařízení a musí být schopné fungovat s nimi zároveň
- datové přenosy musí být dostatečně zabezpečeny
- senzory musí být připojitelné ke službám, které zpracovávají jimi odesílaná data – v některých případech jde např. o připojení do sítě Internet a vzdálené monitorování lékařem

Pro kontinuální měření koncentrace glukózy jsou tyto požadavky také směrodatné. Z pohledu prvního bodu jde o technologie, které nejsou proprietární. Takovými technologiemi jsou například standardy definované v IEEE 802.11 (Wireless LAN[4], dále jen Wi-Fi), IEEE 802.15.1 (Wireless PAN[5], dále jen Bluetooth), IEEE 802.15.4 (Low-Rate Wireless PAN) a protokoly na těchto standardech založené – například nízkopříkonový Bluetooth (Bluetooth Low-Energy) nebo ZigBee.

Z vypsaných protokolů technologie Wi-Fi není v základu stavěná pro nízkou spotřebu. Ačkoliv splňuje ostatní požadavky, není pro kontinuální měření používána.

Oproti tomu jednou ze základních vlastností standardu ZigBee je právě nízká spotřeba. V systémech kontinuálního měření koncentrace glukózy však také

[4]Local area network, místní síť

[5]Personal area network, síť v osobním prostoru

není používán, a to proto, že není nijak více rozšířen v jiných systémech (aby bylo možné propojení např. s mobilním telefonem) a jednak dokáže data přenášet rychlostí pouze 250 kbps. Data se pak přenáší až čtyřikrát déle než například u nízkopříkonového Bluetooth, a to implikuje nutnost mít rádiový modul aktivní po čtyřikrát delší dobu. Z toho je zřejmé, že spotřeba elektrické energie plynoucí z využívání rádiových přenosů bude proporciálně vyšší.

V dostupných senzorech jsou zabudovány protokoly Bluetooth a Bluetooth Low-Energy[25], jelikož splňují všechny výše uvedené požadavky.

4.2.2 Bluetooth

Technologie Bluetooth je definována ve standardu IEEE 802.15.1. Tento standard definuje přenos dat v tzv. bezdrátových PAN sítích, tedy sítích které zahrnují zařízení operující v osobním prostoru člověka. Standardu Bluetooth existuje několik verzí, tato podkapitola se bude věnovat standardu do verze 3.0 a části standardu Bluetooth Classic, jak je označován ve vyšších verzích.

Bluetooth využívá bezlicenční pásmo 2.4 Ghz (2.402 - 2.480 Ghz) a je schopen přenášet data rychlostí až 24 Mbps. Tato technologie je běžně dostupná v mobilních telefonech již po dlouhou dobu. V medicínských aplikacích ji lze vidět například u některých glukometrů, osobních vah, teploměrů a dalších obdobných zařízení. Problematikou aplikace technologie Bluetooth do medicínských přístrojů se zabývá standard IEEE 11073 – *Personal health device communication* [2]. Konkrétní formou zařízení pro kontinuální monitoraci glukózy se zabývá standard IEEE 11073-10425 – *Device specialization – Continuous Glucose Monitor (CGM)*[1]. Standardy pro tuto technologii zaštit'uje uskupení *Bluetooth SIG*[6]. To je uskupením desítek subjektů po celém světě, které tuto technologii využívá ve svých aplikacích a formuje standard pro potřeby moderních technologií a trendů.

Jedna základní síť může obsahovat nejvýše 1 hlavní stanici (*master*) a 7 podřízených stanic (*slave*). Taková síť se nazývá *piconet*. Pokud je vyžadováno propojení více stanic, je možné, aby alespoň jedno ze zařízení bylo připojeno do více sítí typu *piconet* a utvořilo síť nazývanou *scatternet*.

[6]Special Interest Group, lze přeložit jako „uskupení specifických zájmů“

Z pohledu požadavků na medicínské aplikace je Bluetooth vyhovující technologií – interoperabilita je zajištěna jednak integrací do drtivé většiny mobilních zařízení, otevřeným standardem a například i velkou podporou v podobě vývojových knihoven jednotlivých výrobců. Přizpůsobení medicínským aplikacím zajišťují zmíněné standardy.

Kompatibilita s ostatními technologiemi je zajištěna jednak výběrem vysílacích kanálů tak, aby nebyly v konfliktu s ostatními technologiemi operujícími ve stejném frekvenčním pásmu (Wi-Fi, ZigBee) a jednak schématem přeskakování mezi kanály při vysílání (*FHSS*[7]).

Zabezpečení komunikace je umožněno (ne však vynuceno) standardem. Až samotné zařízení definuje úroveň zabezpečení, kterou pro daný komunikační kanál vyžaduje. Může být vyžadováno zabezpečení na úrovni služby (*service-enforced*) nebo linkové vrstvy (*link level-enforced*)[26]. Standard definuje celkem 4 úrovně zabezpečení, které jsou různými kombinacemi zmíněných modelů. Nejnižší úroveň nepoužívá zabezpečení žádné a přenos dat je tedy odposlechnutelný. Pro medicínské aplikace je dále zajímavé schéma, které vyžaduje šifrování. Tím jsou všechny vyšší definované úrovně.

Propojitelnost se službami pro zpracování a vyhodnocení dat je zajištěna tím, že zařízení, ke kterému je vysílač připojen, buď samo zpracovává data nebo obsahuje technologie, díky kterým je možné komunikovat s libovolným zařízením v síti Internet.

Nízká spotřeba je asi jediným kritériem, které je diskutabilní. Běžný modul implementující Bluetooth potřebuje pro vysílání a příjem proud až 150 mA. Je tedy potřeba při návrhu aplikačního protokolu dbát na správnou implementaci standardu a správné řízení spotřeby. To zahrnuje například schéma vypínání modulu na dobu, kdy není potřeba komunikovat. Také je nutné osadit baterii o dostatečné kapacitě, aby zařízení bylo schopné na jedno nabití vydržet alespoň tak dlouho, jako je životnost jednoho senzoru.

Z důvodů nízké spotřeby byl navržen standard Bluetooth Low-Energy popisovaný v následující kapitole.

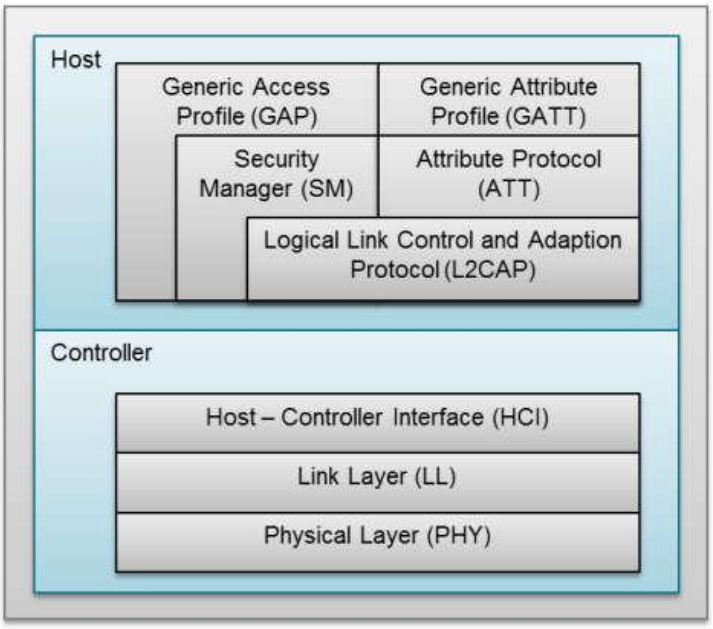
[7]Frequency Hopping Spread Spectrum, metoda přenosu, při které je dle pseudonáhodného klíče prováděno přeskakování mezi komunikačními kanály frekvenčního pásma

4.2.3 Nízkopříkonový Bluetooth

S verzí Bluetooth standardu 4.0 přišlo rozdělení na několik dílčích standardů: Bluetooth Classic (původní standard), Bluetooth High-Speed (vysokorychlostní přenosy) a Bluetooth Low-Energy. Z pohledu zařízení použitelných pro monitoraci glukózy je zajímavý zejména Bluetooth Low-Energy, tedy nízkopříkonový Bluetooth.

Operuje taktéž v bezlicenčním frekvenčním pásmu 2.4 Ghz a maximální přenosovou rychlostí je 1 Mbps. Oproti klasickému Bluetooth standard definuje i řízení spotřeby, kdy je komunikační modul po většinu času nečinný a tedy je možné ho vypnout. Změny na fyzické a linkové vrstvě dovolují redukovat maximální spotřebu na hodnotu okolo $15mA$, tedy cca desetinu původní hodnoty[17]. Dále lze průměrnou spotřebu omezit v rámci implementace aplikačního protokolu, a to až na padesátinu či setinu.

Z pohledu architektury síťového zásobníku je zavedeno dělení na tři části – aplikační část, hostitelská část a řídicí část. Na obrázku 4.2 lze vidět rozdělení hostitelské a řídicí vrstvy na dílčí části. Mezi těmito částmi je definováno rozhraní označované jako HCI[8].



Obrázek 4.2: Bluetooth Low-Energy protokolový zásobník, dělení hostitelské a řídicí části. Obrázek převzat z [5]

[8]Host-Controller Interface, rozhraní mezi hostitelskou a řídicí částí

Řídicí část obsahuje fyzickou a linkovou část standardu. Hostitelská část je dělena na několik podčástí:

- řízení přístupu k médiu a adaptace (L2CAP)
- správa zabezpečení (SM)
- protokol pro správu atributů (ATT)
- profil jednotného přístupu (GAP)
- profil jednotných atributů (GATT)

Část L2CAP se zabývá zapouzdřením dat a předáváním do vyšších vrstev hostitelské části. Také obstarává správnou segmentaci zpráv, aby bylo možné je přenášet po vrstvách nižších. Správa zabezpečení se stará o výměnu klíčů, šifrování, ověřování a autorizaci.

Vrstva GAP přímo komunikuje s aplikační vrstvou a zaštiťuje zjišťování nebo oznamování dostupných služeb a profilů pomocí oznamovacích paketů. Z pohledu GAP existuje dělení zařízení na několik typů:

- broadcaster – pouze vysílá oznamovací pakety do okolí, nepodporuje spojení
- observer – pouze přijímá oznamovací pakety (skenuje), a to buď pasivně, nebo aktivně vyzýváním okolních zařízení
- central – nevysílá oznamovací pakety, pouze je přijímá a při zjištění zařízení, které podporuje spojení umožňuje navázání spojení. Takovým zařízením je například mobilní telefon nebo laptop.
- peripheral – vysílá oznamovací pakety a čeká na požadavky o vytvoření spojení. Typicky poskytuje užitečná data. Takovým zařízením je třeba senzor, měřicí stanice, apod.

Vrstva ATT definuje rozhraní pro přístup k datům. Každá zpřístupněná jednotka má vlastní identifikátor v podobě tzv. UUID[23]. Rovněž je definován způsob přístupování k této jednotce podobou zpráv, které jednotka podporuje:

- Read request – požadavek čtení atributu
- Write request – požadavek zápisu atributu, potvrzovaný
- Write no response – nepotvrzovaný požadavek zápisu atributu
- Indication – potvrzovaná data, jejichž iniciátorem je zdroj dat (server)
- Notification – nepotvrzovaná data, iniciátorem je rovněž zdroj (server)

Vrstva GATT pak umožňuje definovat konkrétní datové jednotky. Atributy sdružuje do tzv. charakteristik, a ty pak do služeb. Služby jsou pak typicky sdruženy ještě do tzv. profilu.

Každá charakteristika má atributy, které definují, jakým způsobem se má k datům přistupovat – zda je potřeba ověření, které zprávy charakteristika podporuje, zda vyžaduje šifrování, apod. Vždy má charakteristika hlavní atribut, kterým je vlastní hodnota, tedy zpřístupňovaná data. Dále může mít definované tzv. deskriptory, které slouží k upřesnění vlastností charakteristiky, mohou obsahovat její jméno v textové podobě a další informace.

Charakteristiky jsou sdružovány do služeb a ty pak do ucelených profilů, které mohou být zaštítěny standardem. Pro kontinuální měření glukózy jde o specializaci s přiřazeným číslem `0x181F`[15].

Profil pro kontinuální monitoraci glukózy

Tento profil deklaruje tyto hlavní charakteristiky:

- CGM Measurement – naměřená koncentrace glukózy
- CGM Feature – obsahuje vlastnosti senzoru a vysílače, konstantní data pro jedno zařízení
- CGM Status – aktuální stav senzoru a vysílače
- CGM Session Start Time – časová značka počátku měření
- CGM Session Run Time – očekávaná celková doba měření, podle této hodnoty lze dopočítat, kdy bude nutné senzor vyměnit

- Record Access Control Point – řídicí charakteristika pro řízené odesílání měřených hodnot např. z minulosti; výstupy jsou zpřístupněny skrze charakteristiku *CGM Measurement*
- CGM Specific Ops Control Point – charakteristika pro kalibraci a další nastavení specifická pro kontinuální měření

Formát dat, která jsou v tomto profilu zasílána, je definován standardem IEEE 11073[2]. Tento standard definuje kromě standardních celočíselných typů i například speciální formát čísla s plovoucí desetinnou tečkou k odstínění závislosti na architektuře.

Z tohoto profilu je pro tuto práci relevantní jednoznačně charakteristika *CGM Measurement*, která obsahuje měřená data buď aktuální, nebo vyžádaná skrze zápis do Record Access Control Point charakteristiky (dále jen RACP). Zápisem do RACP lze vyžádat hodnotu aktuální, libovolnou z minulosti (z určitého okénka, které si senzor pamatuje), všechny z minulosti, nebo lze hodnoty mazat.

Důležitá je také hodnota charakteristiky *CGM Specific Ops Control Point*, jelikož zápisem lze provádět kalibraci senzoru, nastavovat periodu komunikace, nastavovat hodnoty koncentrace, při kterých senzor vyvolá alarm (např. při nízké nebo vysoké hodnotě, při nebezpečné tendenci, apod.).

Zbytek profilu a detaily s ním spojenými lze najít v oficiální specifikaci [15].

5 Zpracování

Přenesená data je třeba analyzovat a výsledek použít pro informování pacienta nebo aktivní léčbu v podobě dávkování inzulinu. Výrobcem senzorů jsou často dodávána zařízení, která naměřené hodnoty zobrazují a ukládají. Po příchodu pacienta k lékaři pak typicky dojde k extrakci dat z tohoto zařízení a vizualizaci naměřených hodnot v časovém diagramu.

5.1 Inzulinová pumpa

U některých případů diabetu může lékař doporučit používání inzulinové pumpy. To je přístroj, který na základě manuálního nebo automatického podnětu vygeneruje potřebnou dávku inzulinu a vpraví ji pacientovi přímo do těla.

Pro automatické podněty je nutné mít zaveden také měřicí přístroj, který kontinuálně měří koncentraci glukózy a v případě příliš vysoké koncentrace tento podnět vygeneruje.

Tato pumpa na základě naměřených dat vyhodnocuje tendence, a pokud je stoupání koncentrace příliš velké, pumpa reaguje nastavenou akcí – upozorněním nebo dávkováním inzulinu.

5.2 Mobilní telefon

Vzhledem ke skutečnosti, že prakticky všechny dostupné senzory pro kontinuální měření obsahují vysílač s technologií Bluetooth nebo Bluetooth Low-Energy, je možné použít i mobilní telefon k připojení senzoru a příjmu dat – ty moderní s touto technologií pracovat umí. Typický mobilní telefon sice nebude vybaven systémem pro dávkování inzulinu, může však obsahovat výpočetní model pro stanovení koncentrace glukózy v krvi, predikci této hodnoty, může data ukládat a může být připojen k dalším zařízením v osobní síti nebo v síti Internet.

Mobilní telefon s operačním systémem Android nebo iOS dnes vlastní podstatná část populace ve vyspělých zemích[34], a tak se případné pořizovací náklady nijak nezvedají.

Jedním z možných problémů je, že ač jsou mobilní telefony obecně považovány za zařízení s nízkou spotřebou, je nutné je poměrně často nabíjet. Tato skutečnost je dána tendencí poslední doby používat mobilní telefon k provozování široké škály aplikací pro přístup k velkému množství služeb.

Dalším možným problémem může být výkon. Pro příjem a zobrazení hodnot ze senzoru to problém není, ale při uvažování modelu a výpočtu jeho parametrů už se jedná o výpočetně náročný problém, který mobilní telefon může počítat delší dobu. Tato doba, kdy je procesorové jádro využíváno na dostupné maximum, se nutně projeví i na spotřebě energie. Výpočet je nutné optimalizovat tak, aby využíval buď omezené prostředky, nebo byl přizpůsoben pro běh na zařízení s omezeným výkonem a dostupnou kapacitou baterie.

Omezení prostředků nemusí být vždy vhodné – v případě využití například jen jednoho procesorového jádra namísto čtyř dostupných může omezit spotřebu, ale zároveň hrozí, že výpočet bude trvat řádově delší dobu. Pak by byla průměrná spotřeba velmi podobná. Mnohem větší smysl může dávat přizpůsobení algoritmu pro běh na takto omezených zařízeních, a to například zvětšením tolerance chyby, respektive snížení počtu iterací (resp. generací) příslušného algoritmu, který může být implementován (viz kapitola 3.4). Snížení počtu iterací nejen že zvyšuje průměrnou chybu, ale také může v případě iteračních nebo evolučních strategií znatelně zvýšit rozptyl chyby, což nemusí být tolerovatelné.

Výsledná aplikace musí vykazovat minimální průměrnou i nárazovou spotřebu. Průměrnou z důvodu dlouhodobého hlediska, aby nedošlo ke snížení celkové životnosti zařízení. Nárazovou pak z důvodu, aby nebyl uživatel „překvapen“ z rychlého úbytku stavu nabití baterie a mohl správně plánovat nabíjení. I to je z pohledu komfortu používání mobilních aplikací a potažmo celého telefonu důležité.

Mobilní telefony s operačními systémy Android a iOS dovolují provádět měření procesorového času přiděleného dané aplikaci[24]. Android k tomu využívá buď integrovaný pohled v rámci nástrojů přímo v telefonu nebo dovede generovat hlášení o spotřebě nástrojem `adb`. Operační systém iOS dovoluje pořizovat „nahrávky“ spotřeby, a to buď přímo v telefonu v nástrojích pro

vývojáře, nebo v rámci prostředí Xcode s připojeným mobilním telefonem k počítači. Tyto nástroje je vhodné využít při vývoji aplikace pro optimalizaci dle kritéria minimální spotřeby.

V současné době existuje několik aplikací, které monitoraci glukózy obstarávají. Vždy jde o specializovaný software výrobců senzorů nebo třetích stran. Například výrobce senzorů Medtronic Inc. vyvinul pro operační systém iOS aplikaci Guardian Connect, která komunikuje se senzorem, ukazuje aktuální hodnotu koncentrace a graf s hodnotami za poslední dobu. Také upozorňuje uživatele v případě potenciálně nebezpečných situací, jako je příliš vysoké stoupání nebo klesání koncentrace glukózy. Obdobnou aplikaci dodává firma Dexcom k senzorům vlastní výroby, a to navíc i pro operační systém Android.

Dále existuje systém s názvem Nightscout[1], který využívá již existujících řešení výrobců senzorů, specializovaného hardware pro překódování výstupů senzorů nebo implementuje protokoly vybraných vysílačů. Hodnoty jednak zobrazuje v mobilním telefonu, a jednak přidává možnost ukládat je v cloudovém prostředí, případně monitorovat koncentraci glukózy na dálku, což se může hodit například pro rodiče dětí s diabetem.

5.3 „Chytré“ hodinky

Do moderního trendu nositelných zařízení patří například i tzv. „chytré“ hodinky. Jde o zařízení nošené na zápěstí jako klasické hodinky, pouze s tím rozdílem, že je na nich nahraný buď operační systém WearOS (Google), watchOS (Apple) nebo některý z uzavřených systémů dalších výrobců. Role chytrých hodinek je často pouze doplňková – jsou propojeny s mobilním telefonem, zobrazují upozornění, která jsou v něm vyvolána a mohou zobrazovat i další vybraná data, která telefon poskytuje.

Uzavřené systémy hodinek např. FitBit nebo Pebble nedovolují zobrazování vlastních dat, jelikož v návrhu je počítáno pouze s takovou množinou funkcí, kterou dodá sám výrobce. Z pohledu modularity je proto vhodné volit hodinky podporující systém WearOS nebo watchOS. Oba tyto systémy podporují kromě instalace dodatečných aplikací i zobrazování vlastních dat prostřednictvím systému zvaného *complications*. Jde o jednotný mechanismus zobrazování datových proudů, jejichž zdrojem je typicky mobilní tele-

[1] http://www.nightscout.info/

fon a aplikace v něm. To umožňuje například zobrazování aktuální hodnoty koncentrace glukózy v podkoží, hodnot poskytnutých výpočetním modelem a tak dále.

Tyto systémy dále obvykle definují omezení nebo doporučení týkající se přidělených prostředků, zde ve formě definice periody získávání dat z datových proudů. Pro zobrazování těchto hodnot postačí perioda měřicího přístroje, která se u CGMS zařízení pohybuje v jednotkách minut – obvykle 5 minut.

V neposlední řadě umí hodinky také spouštět plnohodnotné aplikace, které jsou pouze omezené ovládacími prvky a výkonem přístroje. Vzhledem k malému dotykovému displeji a absenci tlačítek pro uživatelský vstup je nutné omezit se na velmi malé množství informací a správnou agregaci dat. Pro monitoraci glukózy lze uvažovat nad malým grafem, zobrazujícím například poslední hodinu naměřených a vypočtených hodnot. Také je možné integrovat rychlý přístup k vybraným funkcím, kterými u jiných aplikací bývá zpravidla vypínání upozornění na omezenou dobu a další.

5.4 Osobní počítač

Další možností je využití osobního počítače pro příjem a zpracování dat. Toto řešení sice poskytuje největší možný výpočetní výkon a nejmenší nutnost brát ohled na spotřebu, ale není vhodným kandidátem pro přímou monitoraci glukózy. Toto řešení by bez použití mezičlánku výrazně redukovalo mobilitiu uživatele. Může ale plnit funkci doplňkovou k mobilnímu telefonu, který je připojený k síti Internet. Ten může odesílat data ke zpracování do vzdálené stanice, čímž může dojít k výraznému redukování požadavků na výpočetní výkon a elektrickou energii na straně telefonu. Tento princip se označuje jako *offloading*.

Mobilní telefon pak působí pouze jako prostředník k odesílání dat a zobrazování naměřených a vypočtených hodnot. Nutným požadavkem je však stabilní připojení k síti Internet a případné záložní řešení v podobě lokálního zpracování dat. Také je nutné mít po celou dobu monitorace dostupný počítač v síti, který bude data zpracovávat – tím může být buď počítač samotného uživatele, lépe však služba v síti Internet poskytovaná důvěryhodnou stranou. V obou případech je nutné dbát zvýšenou pozornost bezpečnosti komunikačního kanálu a vzdáleného prostředí, jelikož se jedná o citlivá medicínská data.

5.5 Shrnutí

V předchozí kapitole byly popsány dostupné měřicí přístroje a techniky pro běžné „domácí“ měření koncentrace glukózy. Pro potřeby této práce bude dále počítáno s kontinuálním měřením v intersticiální tekutině (senzor zavedený v podkoží) a sporadickým měřením v krvi (glukometr).

Dále byla část věnována přenosovým technologiím, které senzory využívají pro přenos dat do dalších zařízení. Z těchto technologií bude pro tuto práci relevantní pouze technologie Bluetooth a zejména pak Bluetooth Low-Energy, jelikož obě tyto technologie jsou součástí vysílačů všech běžně dostupných řešení.

V této kapitole byly popsány přístroje, které v systému plní roli zpracování dat a poskytnutí upozornění nebo aktivní regulace. Tato práce se dále bude zabývat použitím mobilního telefonu a osobního počítače. Mobilní telefon bude v této práci plnit roli koncového zařízení u pacienta, které ho bude informovat o koncentraci glukózy v intersticiální tekutině a užitím výpočetních modelů poskytovat relevantní informaci o koncentraci glukózy v krvi. Osobní počítač je pak v práci zahrnut z jiného důvodu – vzhledem k požadavku na použití stejného prostředí a kódu pro uživatelské a výzkumné (simulační) účely je v návrhu systému zohledněn stejný základ, který bude možné provozovat na obou zmíněných zařízeních.

6 Simulace

Jedním z požadavků na výsledné řešení je i podpora simulací pro výzkumné účely. Je nutné, aby bylo umožněno přehrávání již naměřených dat, interaktivně zasahovat do řízení simulace a výsledky mít přehledně sumarizované v uživatelském rozhraní. Výsledné řešení musí být ale beze změny implementace možné aplikovat i na reálné měření, a tak je nutné zvolit správnou architekturu.

Hlavním prvkem simulace je kontrakce časů, což reálné měření z podstaty neumožňuje. Máme-li naměřená data s časovými značkami, architektura musí umět zaznamenané hodnoty zpracovávat jak bez kontrakce, tak s kontrakcí, a to buď na pevně daný časový úsek s možností krokování nebo data nezpožďovat vůbec.

Z klasických simulací lze dále zmínit princip kalendáře, tedy plánovacího mechanismu s vnitřní prioritní frontou pro vzniklé události, které jsou řazeny dle času. Vpředu fronty je vždy událost nebo objekt, který vyžaduje obsluhu jako první. Zde musí kalendářem být entita, která je konfigurovatelná – pokud je nastavena kontrakce časů na pevnou periodu, musí zpožďovat hodnoty na výstupu do doby, než uplyne nastavená perioda. Pokud je kontrakce časů vypnutá, musí je zpožďovat (synchronizovat) na reálný čas.

Zde uvažujme dvě části systému – část vstupní a část výstupní. Vstupní část data buď přijímá z reálného senzoru, načítá je z databáze nebo je například generuje dle nějakého klíče. Data odtud budou označena časovou značkou, která vyjadřuje čas, kdy mají být data k dispozici. Výstupní část pouze přijímá tato data od entity kalendáře, a to buď s definovaným zpožděním (pevná perioda nebo reálný čas), nebo okamžitě.

V rámci návrhu systému je dále potřeba dělit vstupní a výstupní část na další oddělené entity, z nichž každá má svou specifickou funkci. Entity mezi sebou musí umět komunikovat – předávat data. Kalendář zde bude hrát roli pouze v řízení komunikace mezi vstupní a výstupní částí.

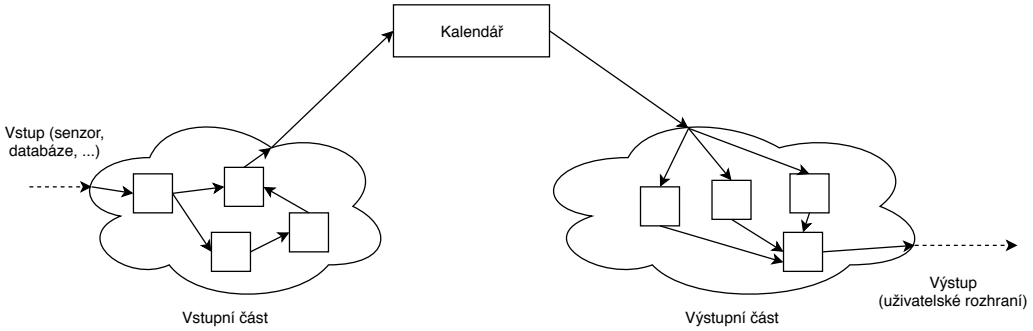
6.1 Otevřená architektura

Uvažujeme-li obecnou architekturu bez zvláštního uspořádání, je každá entita samostatným objektem simulace. Musí definovat propojení mezi entitami, aby bylo možné předávat data, například pro nalezení parametrů modelu, vypočtení signálu na základě těchto parametrů, stanovení chybových metrik a další.

Jelikož jde o architekturu, kde mezi sebou asynchronně komunikují entity bez globální synchronizace, je nutné zavést logický čas. Zde nelze uvažovat jednoduchou časovou značku dle Lamporta, architektura vyžaduje zavedení vektorového času. Tento vektor bude vždy o velikosti N, kde N je počet objektů simulace.

Propojení entit musí vždy zahrnovat i entitu výstupní pro formátování výstupů do uživatelského rozhraní – propojení musí být definováno ve hvězdicové topologii, kdy všechny entity produkující zobrazitelná data definují propojení s výstupní entitou. Dále je nutné uvažovat, že vstupní data musí být odeslána do všech výpočetních entit, které s nimi pracují.

Výše vypsané skutečnosti povedou k relativně vysoké složitosti architektury s velkým množstvím propojení. Potenciálně se zvýší režie a paměťové nároky vlivem duplikace dat po různých propojeních a celkově bude míra redundance dat zbytečně vysoká. Na druhou stranu architektura dovoluje bez větší složitosti distribuovat jednotlivé entity na různá zařízení v distribuovaném systému, tedy rozložení zátěže výpočtů na různé uzly.



Obrázek 6.1: Znázornění otevřené architektury s kalendářem, který řídí komunikaci mezi vstupní a výstupní částí.

6.2 High-Level Architecture

Mezi simulační architektury s širokou škálou aplikací patří i *High-Level Architecture* (dále jen zkratkou HLA), která je standardizována v rámci IEEE 1516. Jedná se o architekturu v základu stavěnou pro distribuované simulace. Definuje objekty simulace, rozhraní a sadu pravidel, kterými se musí objekty řídit[32, 35].

Základem architektury je tzv. *federát*. To je zde označení pro objekt simulace, který má vlastní oddělenou logiku a s ostatními federáty komunikuje výhradně prostřednictvím *běhového rozhraní* (dále jen RTI[1]). Federát se musí připojit do simulace, pokud chce s jinými federáty komunikovat. Jeden federát se může připojit do více simulací, ale i do stejné simulace víckrát. Simulace jako skupina federátů se zde nazývá *federace*.

V rámci jedné federace je dodržován jednotný formát zpráv a komunikační schéma. Tato sada pravidel je nazývána *federation object model* (dále jen FOM) a využívá schéma definice zvané *object model template* (dále jen OMT).

Další důležitou komponentou HLA je již zmíněné běhové rozhraní. To zde zastává funkci prostředníka pro komunikaci všech federátů. Definuje jednotné rozhraní (dané standardem), přes které jsou jednotlivými federáty zasílána data. I přesto, že RTI je důležitým prvkem celé architektury, je různými výrobci implementováno jinak, a to má za důsledek nekompatibilitu mezi výrobci.

Komunikace mezi federáty probíhá vždy skrze RTI. Jednotlivé datové jednotky jsou objekty (např. serializované nebo jiným způsobem převedené do přenositelné formy). RTI používá model komunikace obecně označovaný jako *publish/subscribe* – jednotlivé federáty se musí zaregistrovat (*subscribe*) pro příjem konkrétního datového toku (objekty daného formátu) vytvářeného a odesílaného (*publish*) z federátu jiného.

Oproti obecné otevřené architektuře má HLA rozhodně výhodu ve standardizaci rozhraní a možnosti distribuovat jednotlivé federáty do různých uzlů v síti[35]. Používá model komunikace *publish/subscribe* s prostředníkem, takže není nutné, aby každý federát věděl o všech ostatních, pouze musí být známá společná množina informací, jako jsou struktury zpráv a

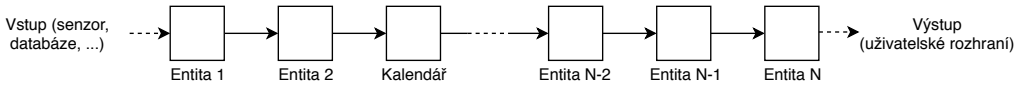
[1] z originálu: *runtime interface*

existující datové proudy. Architektura však nabývá na složitosti a pro potřeby řešeného problému je příliš složitá. V rámci mobilní aplikace by navíc přinesla zbytečnou režii a vlivem zvýšené bezpečnosti operačních systémů pro mobilní telefony by ji bylo nutné značně degradovat.

6.3 Lineární architektura

Lineární architekturou rozumíme takovou architekturu, kde jsou jednotlivé entity mezi sebou propojeny nejvýše jednou vazbou vždy v lineární topologii. Topologie je orientovaná a propojení jsou vždy pouze jednosměrná, lze je tedy označit za simplexní.

Entity mezi sebou komunikují vždy užitím zpráv, které kromě typu zprávy a dat obsahují i zmíněnou časovou značku. Entity pak zprávy přijímají, vyhodnocují a na výstupu mohou generovat odpovídající odpověď, která je odeslána do entity následující.



Obrázek 6.2: Znázornění lineární architektury

6.3.1 Fall-through architektura

Z pohledu problému je zajímavá architektura, která by se dala označit jako *fall-through*, jež je také lineární. Předpokládá, že na začátku lineární topologie existuje entita generující zprávy. Ta je předává dalším entitám a bez ohledu na to, zda entita zpracovává přijatou zprávu nebo ne, pošle ji další entitě v řetězu. Zpráva tedy vždy „propadává“ skrze celý řetěz a dostane se ke všem entitám. Každá entita při příjmu může zprávu vyjmout, zpracovat a potenciálně ji i zahodit. Tato architektura dále umožňuje kterékoliv entitě asynchronně vůči generátoru zpráv vytvořit zprávu vlastní a do úseku řetězu před ní ji poslat.

Také je možné v této architektuře úplně vynechat logický čas, jelikož jsou zprávy vždy seřazené, nesmí se předbíhat a z pohledu každé entity přijdou vždy v takovém pořadí, v jakém vznikaly. Řazení zpráv je tedy zařízeno implicitně. Pokud by realizace vyžadovala přenos části řetězu na jiný uzel

(např. kvůli výpočtům), musí existovat entita, která bude tuto funkcionalitu na obou částech zajišťovat. Také je kvůli absenci časových značek nutné použít takový transportní protokol, který zaručuje doručení zpráv v pořadí, ve kterém byly vyslány – například protokol TCP.

Kalendářem zde může být za daných okolností jedna samostatná entita řetězu, jelikož pouze řídí časové zpoždění (resp. synchronizaci) dat ze vstupu.

7 Návrh systému

Požadavky na výsledný systém jsou:

- vstup dat do architektury
  - z reálného senzoru
  - z databáze
  - ze souboru (extrakce ze známých formátů)
- zpracování dat na různých platformách užitím jedné implementace
  - mobilní telefon
  - osobní počítač
- schopnost fungovat v reálném čase (skutečné měření)
- podpora simulací (výzkumné účely)
- výpočet koncentrace glukózy v krvi
  - modely
    - model Steil-Rebrinové
    - difúzní model
  - výpočet parametrů
    - NEWUOA (knihovna NLopt)
    - metadiferenciální evoluce
- vykreslení grafů a chybových mřížek
  - v plné míře na osobním počítači
  - v omezené míře na mobilním telefonu
- aplikace pro osobní počítač s použitím knihovny Qt
- výpočet chybových metrik pro určení kvality modelu a jeho parametrů
- modularita řešení v podobě dynamických knihoven
- implementace co největší části v jazyce C++ s použitím moderního standardu C++17

7.1 Modularita

Vzhledem k tomu, že systém se skládá z oddělených modulů, z nichž každý má svou specifickou funkci, lze zajistit jejich oddělení do dynamických knihoven vhodnou definicí rozhraní. Aby bylo dosaženo plné modularity, je vhodné uvažovat i nad automatizací řešení načítání knihoven. Dynamické knihovny ze své podstaty exportují symboly, které jsou přístupné z vnější části (např. z programu, který je načítá).

Před návrhem rozhraní je nutné definovat třídy funkcionality, tedy typy jednotlivých entit, které v celém systému figurují. Modul, který obstarává hlavní funkcionalitu v podobě výpočtů, načítání, vykreslování a dalších obdobných součástí, a je tedy plnohodnotnou entitou architektury (je zapojen do lineární topologie) nazvěme *filtr*. Tento filtr bude na vstupu zprávy přijímat a na výstupu je předávat dál, volitelně pak může asynchronně vůči vstupní části generovat zprávy další a posílat je na výstup. Kalendář bude také implementován jako filtrů.

Dalším typem modulu musí být model dynamiky glukózy. V požadavku je sice podpora pouze dvou modelů, ale vzhledem k tomu, že do budoucna je počítáno s rozvojem stávajících a vývojem nových, je nutné je modularizovat. Kromě modelů musí samostatnou entitou být i výpočetní algoritmus, respektive kód, který se stará o transformaci jednotně formátovaných požadavků a algoritmus spouští. Důvod je opět stejný – NEWUOA a metadiferenciální evoluce nejsou jedinými algoritmy, které vedou k nalezení řešení, a do budoucna je nutné počítat s možným rozšířením o další.

Oddělit je ale nutné i jednotlivé podpůrné funkce, jako je výpočet metriky a metoda aproximace (resp. interpolace) signálu. Měření je sice označováno jako kontinuální, ve skutečnosti jde však o naměřené hodnoty v několikaminutových intervalech. Mezilehlé části je nutné dopočítat, a stejně jako u algoritmů i modelů, i zde existuje velké množství metod aproximace a interpolace, které mohou mít odlišné vlastnosti. Metriky taktéž vyžadují modularizaci z velmi podobného důvodu – zde navíc jde i o kritérium optimalizace, takže kromě přesnosti modelu může být vyžadováno upřednostnění některé z metrik (např. minimalizace maximální nebo průměrné chyby, apod.).

Pro možnost automatizace musí každý modul v rámci dynamické knihovny poskytovat popisovač své funkcionality – jednotnou strukturu pro každou

podporovanou entitu, kterou podporuje. Také musí exportovat tovární funkci, kterou je zapouzdřená entita vytvářena a v případě filtrů spouštěna.

7.2 Přenositelnost

Implementací v jazyce C++ je do značné míry zajištěna i přenositelnost mezi běžnými platformami. Vzhledem k vlastnostem a strukturám, které jsou v jazyce zahrnuty do standardu C++17 včetně, je z velké části zajištěna i minimalizace kódu, který je přímo závislý na platformě. Nutnost psát takto závislý kód ale není eliminována a pro vyrovnání některých odlišností je nutné dopsat funkce, které budou použité rozhraní sjednocovat a implementovat na jednom místě, aby se ve zbytku systému úplně eliminovala nutnost psát kód závislý na platformě a překladači. Vedoucím práce byl doporučen jako jednotný formát rozhraní WinAPI, tedy rozhraní použité v operačních systémech Microsoft Windows.

Jediným problémem v rámci přenositelnosti je uživatelské rozhraní. Požadavky na ovládací prvky a zobrazované informace se diametrálně liší jak z pohledu platformy a technických možností, jako je například velikost displeje a metoda zadávání informací (dotykový displej, klávesnice a myš), tak z pohledu účelu (reálné měření na straně pacienta a simulace na straně vědce). Z tohoto důvodu je nutné přistoupit na implementaci dvou aplikací, které budou obě využívat jako podlehlou výkonnou vrstvu výše popsaný přenositelný systém.

Pro aplikaci pro osobní počítač byla vedoucím práce požadována implementace s použitím knihovny Qt, která na těchto zařízeních zajišťuje přenositelnost i mezi prostředími a operačními systémy (Microsoft Windows, GNU/-Linux, macOS a další). Pro mobilní telefon sice částečná podpora Qt také existuje, z předchozích zkušeností však není příliš vhodná.

Vývoj aplikací pro mobilní telefon může být uskutečněn v prostředí a jazyce přímo doporučeným výrobcem dané platformy, respektive autorem operačního systému – například pro OS Android jde o jazyk Java nebo Kotlin a prostředí Android Studio, pro iOS jde o jazyk Swift (historicky i Objective-C) a prostředí Xcode. To ale implikuje nutnost implementovat více variant aplikace. Proto byly vytvořeny specializované hybridní technologie, které implementují jednotnou „spojku“ pro běh aplikací napsaných v jednom jazyce a prostředí, a samotný překlad nebo interpretace jsou vyřešeny až v momentě

sestavení a nahrávání na dané mobilní zařízení.

Z těchto technologií lze uvést zejména Apache Cordova (PhoneGap), React Native a Xamarin. Apache Cordova je hybridní technologií využívající zapouzdření webové aplikace (HTML5 a Javascript) užitím nativního ovládacího prvku. Další moduly je nutné připojit přes rozhraní webového klienta do kódu hostitelského programu. Tato technologie není vhodná vzhledem k relativně vysoké složitosti propojení nativních knihoven psaných v C++, které by se musely přes JNI[1] nebo nativní rozhraní iOSu připojovat a poté předávat do rozhraní webového. Obdobná situace je u nástroje React Native.

Xamarin je nástroj pro vývoj mobilních aplikací v jazyce C# nad platformou .NET. V rámci balíku Xamarin.Forms je možné definovat i uživatelské rozhraní v jazyce XAML[2] bez nutnosti vytváření mutací pro jednotlivé platformy. Propojení s nativními knihovnami je možné sice také přes rozhraní JNI, zde ale propojení zajišťuje sám Xamarin, takže není nutné duplikovat logiku jak kvůli přenosu přes dvoje rozhraní (JNI nebo nativní rozhraní a .NET), tak kvůli rozdílné platformě (Android, iOS). Přenositelnost kódu je tedy zajištěna mnohem pohodlněji, než u zmíněných nástrojů, které utilizují webové technologie.

7.3 Architektura

Návrh systému bude vystavěn nad simulační architekturou popsanou v předchozí kapitole – *fall-through* architekturou. Jedná se o nejvhodnějšího kandidáta vzhledem k tomu, že by v případě použití jiných architektur celý systém nabyl na složitosti, zvedla by se redundance dat a bez použití lineární topologie by bylo nutné zavést hvězdicovou topologii na vstupní i výstupní části systému kvůli nutnosti propagovat naměřená data do více modulů zároveň.

Každý modul bude tedy tvořit samostatný objekt architektury. Jediným problémem závislým na architektuře je definice rozhraní pro vstup a výstup dat. Vzhledem k faktu, že každý filtr bude mít nejvýše jednu vstupní a nejvýše jednu výstupní vazbu (komunikační kanál), nabízí se několik možností – přímá komunikace voláním exportované funkce, použití prostředníka (např.

[1] Java Native Interface, rozhraní pro komunikaci nativních knihoven s programem běžícím ve virtuálním stroji

[2] Extensible Application Markup Language, jazyk pro popis uživatelských rozhraní platformy .NET

princip *roury*) nebo použití socketů nebo pojmenovaných rour.

Přímá komunikace není vhodná vzhledem k dekompozici – jednotlivé filtry by neměly mít znalost entity na druhém konci, což je kvůli dekompozici do dynamických knihoven problém např. i v otázce správy paměti. Řízení funkce by nabylo na složitosti a oproti jiným modelům je zhoršena i interoperabilita s programy vytvořenými v jiných jazycích a jinými kompilátory. Také by každý filtr musel definovat obsluhu zpráv odděleně a musel by odděleně implementovat frontu, aby nezablokoval filtr předchozí. Použití socketů nebo pojmenovaných rour (lze sjednotit do kategorie BSD socketů) také není příliš vhodným kandidátem, a to z důvodu obtížné správy zdrojů – filtry by musely mít přiřazený komunikační kanál nadřazeným programem, kterým je buď komunikační síťový port nebo soubor na souborovém systému. Síťové porty mohou být již přiřazené nebo používané jiným procesem spuštěným na daném zařízení a navíc je zvýšená režie vlivem využití protokolového zásobníku.

Použití prostředníka pro komunikaci znamená dodatečnou programovou implementaci *roury*, jejíž implementace se dá přizpůsobit tak, aby měla minimální kompatibilní rozhraní (operace *odeslat* a *přijmout*). Tato roura bude vnitřně řešit vkládání do fronty a synchronizaci pro konkurentní přístup. Každý filtr pak při jeho vytváření dostane řídicím programem přidělenou rouru vstupní a výstupní. Dva sousední filtry tedy budou vždy jednu rouru sdílet – jeden ji použije jako výstupní, druhý jako vstupní.

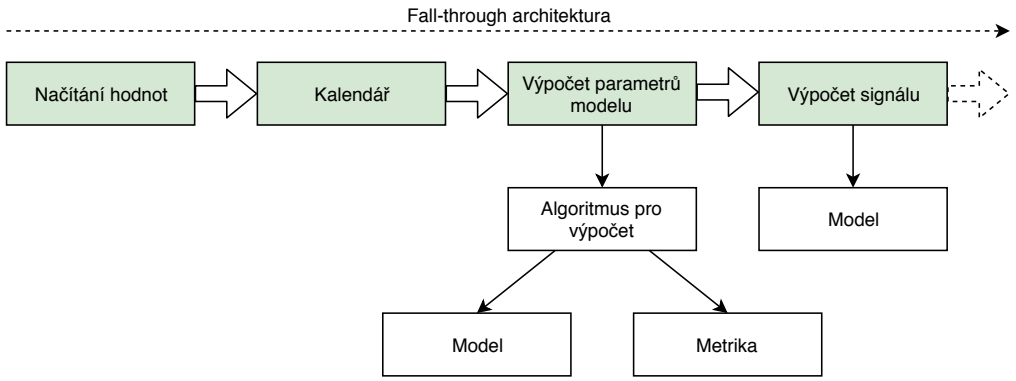
Filtr může také poskytovat dodatečné rozhraní, například pro export vybraných dat do uživatelského rozhraní – například je nesmysl, aby se vykreslený obrázek propagoval skrze roury. Pro tyto účely je možné spoléhat na správnou implementaci a řídicím programem přetypovat obecnou instanci filtru na specifickou, ale to není vhodné zejména proto, že tento přístup neuvažuje možné chyby programátora, a hlavně částečně narušuje princip úplné dekompozice a zavádí „znalost“ specifického objektu do řídicího programu. Z tohoto důvodu je lepší využít principy standardního COM[3] rozhraní, jelikož je to model přímo navržený pro platformně nezávislé komponenty, které spolu dovolují komunikovat. Toto rozhraní definuje metodu `QueryInterface`, kterou se volající dotazuje na konkrétní podporované rozhraní. Pokud je rozhraní podporováno, lze volat jeho metody.

[3]Component Object Model

7.4 Moduly a jejich vazby

Většina jednotlivých funkcí a jejich vazeb je v podstatě dána původním systémem *gpredict2*, který vedoucí práce vyvinul a používá – například vztah modelů, hledání parametrů a metrik je v zásadě daný, jen je potřeba ho přizpůsobit nové architektuře.

Z požadovaných funkcí je pak nutné uvažovat filtry pro načítání hodnot z databáze, souboru a z reálného senzoru. Dále bude potřeba vyvinout modul pro výpočet parametrů modelu. Ten bude navíc oddělen od výpočtu výsledného signálu, jelikož logicky jde o dva rozdílné problémy. Také pro generování výstupů ve formě jednak grafů, a jednak chybových metrik je nutné mít specializované filtry. V neposlední řadě by měl být k dispozici záznam o proběhlé simulaci, což bude zajišťovat také specializovaný filtr.



Obrázek 7.1: Příklad filtrů a vazeb na další moduly. Světle zeleně jsou označeny filtry ve fall-through architektuře

Obrázek 7.1 znázorňuje příklad části systému. Filtr pro výpočet parametrů modelu zde přímo spouští algoritmus pro výpočet, který je parametrizován příslušným modelem a metrikou použitých pro výpočet.

7.5 Simulace reálného zařízení

Pro ověření správné implementace komunikace se senzorem v mobilní aplikaci je nutné buď vlastnit reálný senzor, nebo mít k dispozici takové zařízení, které funkcionalitu senzoru simuluje za použití standardizovaných protokolů. Reálný senzor pro práci poskytnut nebyl, nabízí se proto implementace simu-

lovaného senzoru, která používá zmíněný standard IEEE11073 se specializací na kontinuální měření glukózy.

Tímto simulátorem může být buď implementace v rámci aplikačního software na osobní počítač nebo telefon, nebo na nějaký z vývojových modulů. Nutnou podmínkou je potřebné hardwarové komunikační vybavení – vysílač Bluetooth Low-Energy.

Tento vysílač je sice obsažen ve většině notebooků a mobilních telefonů, prakticky ale používat některé z těchto zařízení implikuje nutnost mít k simulačnímu prostředí navíc ještě další, které bude senzor simulovat. To v případě notebooků a mobilních telefonů není příliš levné řešení. Dostupnějším a praktičtějším se jeví použití vývojových desek, které jsou běžně dostupné, mají minimální nároky na příkon elektrické energie a již obsahují připravené knihovny a nástroje pro vývoj aplikačního protokolu nad zásobníkem Bluetooth Low-Energy.

Jednou z takových desek je například modul firmy Texas Instruments s označením LAUNCHXL-CC2640R2 (případně kterákoliv jiná deska obsahující modul CC2640, CC2650 a další). Tyto desky jsou levné, poskytují stabilní a odladěný Bluetooth zásobník, mají minimální spotřebu a pro aplikační software je použit operační systém reálného času TI-RTOS. Implementace IEEE11073 s profilem pro kontinuální měření glukózy pak obnáší pouze definici příslušných služeb a charakteristik a obslužný software, který bude umět simulovat měření (předávat již v minulosti naměřené hodnoty nahrané do paměti).

7.6 Shrnutí

V této kapitole byl navržen systém splňující požadované parametry. Bude implementována fall-through architektura, v níž budou figurovat výkonné moduly zvané filtry. Tyto filtry budou dále dle specializace využívat další moduly, kterými může být model dynamiky glukózy, algoritmus pro výpočet parametrů modelu, metrika, aproximace a interpolace. Filtry mezi sebou budou komunikovat vlastní implementací roury, která bude řešit konkurentní přístup.

Pro vývoj této architektury bude použit programovací jazyk C++. Pro vývoj aplikace pro osobní počítač bude použita knihovna Qt a programovací

jazyk C++. Pro vývoj mobilní aplikace bude použita technologie Xamarin s knihovnou Xamarin.Forms a programovací jazyk C#.

Dále bude vyvinut simulační software pro vývojovou desku s vysílačem Bluetooth Low-Energy, který bude v daných časových intervalech poskytovat hodnoty naměřené v rámci předchozích měření v minulosti.

8 Implementace

Výsledné programy byly implementovány se společným základem *fall-through* architektury, která je přenositelná mezi platformami. Byla implementována aplikace pro mobilní telefon užitím technologie Xamarin s knihovnou Xamarin.Forms a aplikace pro osobní počítač užitím technologie Qt.

V této kapitole bude popsána implementace obou částí a podlehlé architektury. Také bude popsána implementace jednotlivých entit simulace, zejména pak filtrů, jelikož obstarávají hlavní výkonnou část celého řešení. Rozšířená dokumentace je pak obsažena v příloze A.

Pro vybrané části implementace byla navíc použita knihovna *Intel Threading Building Blocks*, která implementuje paralelizační mechanismy a synchronizační primitiva.

Všechny třídy, které spadají do dané architektury mají definováno rozhraní (identifikátor začíná písmenem I) a odpovídající implementace (identifikátor začíná písmenem C). Také je definováno pojmenování chytrého ukazatele (`std::shared_ptr`) (začíná písmenem S).

Všechny entity jsou identifikovány standardizovaným GUID identifikátorem, který je unikátní pro každý typ.

8.1 Architektura

Jednotlivé entity byly implementovány jako dynamické knihovny příslušné platformy – pro operační systém Microsoft Windows jde o DLL knihovny, pro operační systém GNU/Linux a mobilní operační systém Android na jeho jádře založeném jde o sdílené objekty (*shared object*). Pro mobilní operační systém iOS však musí být knihovny linkovány staticky, jelikož platforma neumožňuje pro uživatelské aplikace použít dynamické linkování.

Komunikace mezi filtry je zajištěna instancemi třídy, které implementují `IFilter_Pipe`. Ta poskytuje rozhraní pro odesílání a příjem zpráv a vnitřně implementuje mechanismus fronty užitím `tbb::concurrent_bounded_queue`. Tato fronta zajišťuje i konkurentní přístup, jelikož filtry mohou přistupovat

do sdílené paměti paralelně. Její implementací je pak třída `CFilter_Pipe`.

Volání `receive()` může být blokující, pokud je vnitřní fronta prázdná. Vzhledem k tomu, že fronta má definovanou maximální velikost, může být blokující i volání `send()`. S tím počítají i jednotlivé filtry – pokud je potřeba provádět specifické operace asynchronně vůči architektuře a datovému toku přes roury, musí spustit další vlákno. Předávání zpráv skrze roury musí navíc být co nejrychlejší, aby byla zvýšena responzivita celého systému.

8.2 Správa paměti

Vzhledem k tomu, že jsou implementace odděleny do dynamických knihoven a zprávy jsou předávany přes rozhraní a zároveň musí být umožněno použití více programovacích jazyků a kompilátorů, musí být definováno i pozměněné schéma správy paměti. V jazyce C++ od standardu C++11 existují chytré ukazatele, které vnitřně provádí počítání odkazů, jehož implementace využívá principů RAII[1]. Tyto ukazatele však nejsou přenositelné mezi programovacími jazyky a jejich implementace ani mezi kompilátory. Bylo proto nutné definovat vlastní správu čítání odkazů, která bude zpřístupněna skrze rozhraní sdílené knihovny.

Zde bylo opět respektováno doporučení vedoucího práce a bylo definováno rozhraní `IReferenced`, které odpovídá rozhraní `IUnknown` z COM WinAPI. Definuje metody pro přičtení a odečtení jednotky od čítače odkazů, a pro zjišťování podporovaných rozhraní. Odpovídající implementací, která je oproti COM výrazně redukována na nutný základ, je třída `CReferenced`. Ta implementuje potřebné metody pro zvýšení a snížení hodnoty čítače. Prvotní vytvoření zvyšuje čítač na jednotku, poslední volání metody pro uvolnění odkazu snižuje hodnotu čítače na nulu a objekt je uvolněn z paměti.

Pro vytváření objektů, které dědí od `CReferenced` je vytvořena šablonová metoda `Manufacture_Object`, která transparentně volá konstruktor třídy, což je umožněno mechanismem variadických šablon standardu jazyka C++11. Žádný podprogram by neměl tyto instance vytvářet přímo (tedy volat `new` nebo inicializovat instance užitím chytrých ukazatelů).

Implementace entit v C++ by pak měla využívat vždy metod pro převod

[1]Resource Acquisition Is Initialization – princip využívající konstruktorů a destruktorů tříd pro získávání a uvolňování zdrojů

těchto objektů na chytré ukazatele jazyka C++. Třída `std::shared_ptr` umožňuje definovat vlastní tzv. *deletor*, tedy funcki, která se volá při skončení platnosti objektu ukazatele, jež by za normálních okolností snížila interní čítač. Ta je zde přepsána na funkci volající vlastní implementaci snížení čítače v rámci třídy `CReferenced`. Pro převod na chytré ukazatele jsou definovány funkce `make_shared_reference` a další.

Také je vyžadována implementace struktur dynamických velikostí, a to ze stejného důvodu – kompatibilita mezi různými jazyky a kompilátory. Proto bylo definováno šablonové rozhraní `IVector_Container` a odpovídající implementace `CVector_Container`. To představuje zjednodušenou implementaci dynamického pole, která je sice vnitřně zajištěna instancí standardní struktury `std::vector`, ale navenek poskytuje kompatibilní rozhraní. Definuje (resp. implementuje) metody `set()` a `get()`, které vždy vyžadují na vstupu dvojitý ukazatel na začátek a konec. Při nastavování nebo získávání hodnoty je pak iterováno od počátku (hodnota ukazatele na začátek) až do konce (hodnota ukazatele o 1 prvek za koncem).

8.3 Filtry

Jednotlivé filtry vždy implementují rozhraní `IFilter`. Toto rozhraní definuje metodu `Run()`, která přejímá jako parametr vektor konfiguračních voleb. Více o konfiguraci filtrů je uvedeno v kapitole 8.5. Filtr je vždy spouštěn ve vlastním hlavním vlákně řídicím kódem. Pokud vyžaduje pro svou funkci asynchronní běh vůči architektuře, může spouštět vlastní vlákno. Vždy je ale povinen toto vlákno ukončit před ukončením vlákna hlavního. Filtru je v konstruktoru předán ukazatel na vstupní a výstupní rouru.

Typickým základem každého filtru je cyklus, který čte z předchozí roury a zapisuje bez modifikace dat do roury následující. V rámci tohoto cyklu definuje vlastní funkcionalitu, kterou může delegovat do odděleného vlákna, pokud existuje možnost, že bude trvat delší dobu.

Filtry si prostřednictvím rour předávají zprávy jednotného formátu. Tuto strukturu lze vidět ve výpisu kódu 8.1. Zprávy jsou identifikované typem, dále obsahují identifikátor zařízení, neseného signálu, časovou značku a identifikátor segmentu. Dále obsahují *union*[2] obsahující nesenou hodnotu – ta

[2]struktura o velikosti největšího prvku, liší se pouze interpretací

může být buď hodnotou naměřeného nebo vypočteného signálu, ukazatelem na kontejner parametrů nebo ukazatel na kontejner s řetězcem.

```
struct TDevice_Event {
        NDevice_Event_Code event_code;
        GUID device_id;
        GUID signal_id;
        double device_time;
        uint64_t segment_id;

        union {
                double level;
                IModel_Parameter_Vector* parameters;
                refcnt::wstr_container* info;
        };
};
```

Výpis kódu 8.1: Struktura zprávy předávané rourami

Dále je pro filtry důležitá organizace naměřených dat. Data jsou dělená do tzv. časových segmentů, které sdružují hodnoty naměřené v rámci jednoho měřicího sezení. Třída segmentu implementuje rozhraní `ITime_Segment`, a to v rámci implementace třídy `CTime_Segment`. Každý segment vnitřně udržuje vektor signálů, které jsou v rámci něj naměřené. Tento signál má rozhraní `ISignal`, implementované ve specializovaných signálech naměřených signálů nebo modelů. Každý naměřený signál je implementován třídou `CMeasured_Signal`, každý vypočtený signál má svou vlastní třídu (např. `CDiffusion_v2_blood`, ...), která dědí od třídy `CCommon_Calculation`.

8.3.1 Vstupní filtry

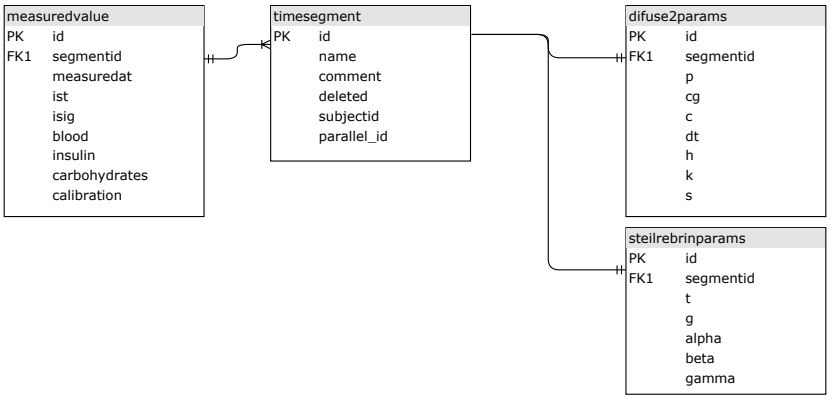
Na vstupu do architektury jsou filtry, které přijímají měřené hodnoty z reálného zařízení, z databáze nebo ze souboru hodnot generovaného měřicím přístrojem.

Filtr, který přejímá hodnoty z reálného zařízení je specifickým filtrem, který pouze přejímá hodnotu na vstupu, transformuje ji na zprávu označenou příslušnou časovou značkou (aktuální čas) a odešle ji do architektury skrze rouru.

Pro potřeby simulace byly implementovány i filtry, které načítají hodnoty z již naměřených dat. Ty převádějí vstupní hodnoty na jednotky *mmol/l*.

Čtení z databáze

Filtr pro čtení z databáze přejímá konfigurační parametry v podobě přístupových údajů do databáze. Struktura databáze je daná z předchozích prací vedoucího práce, tato práce se tedy návrhem schématu nezabývá. Důležitá bude pouze znalost tabulek obsahujících časové segmenty (naměřená data v jednom úseku u jednoho pacienta) a parametry modelů.



Obrázek 8.1: Schéma části databáze podstatné pro tuto práci

Filtr je implementován třídou `CDb_Reader`. Načítá v minulosti naměřená data z databáze dle zadaných identifikátorů segmentů. Časy, ve kterých byly dané hodnoty naměřeny, posouvá do současnosti, aby první naměřená hodnota odpovídala času při začátku simulace. Další hodnoty respektují časové rozestupy mezi měřeními.

Také načítá v minulosti již vypočtené parametry z databáze (pokud již byly vypočteny v minulosti v rámci jiné simulace) a posílá je do architektury skrze roury jako počáteční odhad řešení.

Extrakce ze souboru

Tento filtr částečně přejímá implementaci extrakce ze souborů známých formátů z předchozí práce (portál *diabetes.zcu.cz*). Chová se obdobně jako filtr pro čtení dat z databáze, pouze s rozdílem toho, odkud čerpá naměřená data. Filtr je implementován třídou `CFile_Loader`.

Formát souborů je daný výrobcem senzoru nebo specializovaného přístroje k senzoru dodávaného. Jak bylo zmíněno v kapitole 5, jde o přístroje, které

ukládají naměřené hodnoty, aby je mohl pacient při příchodu k lékaři předat odborníkovi a ten mohl upravit léčbu.

Formáty používají schéma CSV souborů[3], XML[4] nebo XLS (popř. XLSX)[5]. Z těchto formátů jsou vybrány pouze údaje relevantní k problému. Jde pouze o naměřené hodnoty a časy, ve kterých byly hodnoty naměřeny.

8.3.2 Časová synchronizace hodnot

Tento filtr zastává funkci kalendáře z terminologie simulací. Dle konfigurace hodnoty buď synchronizuje na reálný čas nebo zpožďuje na definovanou periodu.

Pro možnost simulaci krokovat je implementována i reakce na specifické zprávy, které notifikují vnitřní podmínkovou proměnnou a aktuálně pozdržovanou hodnotu odešlou skrze rouru okamžitě. Další hodnota je pak zpožděna s respektováním původního časového rozestupu.

8.3.3 Výpočet parametrů modelu

Filtr pro výpočet parametrů modelu, implementovaný třídou `CCompute_Filter`, přijaté hodnoty ukládá do instancí třídy časového segmentu. Hodnoty jsou v rámci něj členěny do jednotlivých signálů, podle kterých se následně řídí modely.

Tento filtr je konfigurován modelem, jehož parametry má počítat, algoritmem pro výpočet parametrů, metrikou a parametry metriky. Také vyžaduje konfiguraci spouštěcí podmínky pro výpočet – tou může být počet naměřených hodnot nebo událost (značka konce časového segmentu nebo příchozí hodnota kalibrace senzoru).

Filtr po výpočtu porovná nové řešení se stávajícím uloženým na stejné množině dat a stejnou metrikou. Pokud je metrika lepší, než metrika řešení doposud považovaného za nejlepší, je asynchronně vygenerována zpráva s novou sadou parametrů daného modelu a odeslána výstupní rourou.

[3]Comma Separated Values – textový soubor s hodnotami oddělenými čárkami nebo středníky

[4]Extensible Markup Language – značkovací formát pro hierarchicky organizovaná data

[5]formát používaný nástrojem Microsoft Excel

8.3.4 Výpočet signálu

Výpočet hodnot signálu je oddělen od výpočtu parametrů do třídy `CCalculate_Filter`. Tento filtr přijímá na vstupu parametry modelu, ukládá je a s každou nově příchozí referenční hodnotou naměřeného signálu vypočte odpovídající hodnotu dle zadaného modelu.

Na počátku počítá s výchozími parametry modelu, postupně přejímá parametry vypočtené filtrem, který se o výpočet stará. Je tedy v architektuře zapojen zpravidla bezprostředně za ním. Tento filtr za normálních okolností vždy počítá hodnoty inkrementálně, nedopočítává hodnoty z minulosti. Reaguje ale na zprávy s požadavkem o přepočtení celého segmentu. Při příchodu takové zprávy vygeneruje zprávu o vymazání všech hodnot z minulosti a dopočítá hodnoty dle aktuálních parametrů modelu. Všechny hodnoty odesílá do výstupní roury.

8.3.5 Vizualizace signálu

Filtr pro vizualizaci signálu opět přejímá velkou část již hotové implementace z předchozího projektu (portál *diabetes.zcu.cz*), pouze byly provedeny drobné úpravy pro kompatibilitu s novým kódem a vykreslovací logika byla obalena filtrem implementovaným třídou `CDrawing_Filter`.

Z přijatých hodnot vypočteného a naměřeného signálu vytváří sadu grafů a jiných pohledů, které jsou relevantní pro monitoraci glukózy, pro výzkumné účely a například i pro porovnávání kvality modelů a signálů z nich vypočtených.

Hlavním grafem je časový vývoj signálů za celé období měření. Dále je generován pohled který tyto hodnoty agreguje do tzv. denního grafu, tedy čáry signálů zalamuje po 24 hodinách. Také generuje Clarkeho a Parkesovu chybovou mřížku, která se používá pro určení kvality modelu. Posledním pohledem je pohled zvaný *ambulatory glucose profile* (nebo zkratkou AGP).

8.3.6 Další filtry

Mezi další filtry patří například filtr, který zaznamenává celý průběh simulace. Ten ukládá záznam do souboru a volitelně i do uživatelského rozhraní.

Dále je přítomen filtr pro přenos zpráv po síti do jiného zařízení – v případě, že by bylo nutné řetěz přerušit a například propojit reálné měření s výstupy v aplikaci na osobním počítači.

V neposlední řadě byly implementovány filtry pro vstup z uživatelského rozhraní (tedy ovládání simulace) a pro zakončování řetězu a uvolňování zpráv z paměti.

8.4 Další entity

Další entity figurující v architektuře byly implementovány do oddělených dynamických knihoven. Vždy mají definované rozhraní pro vytváření a používání.

8.4.1 Modely

Každý model dědí od třídy `CCommon_Calculation`, která implementuje rozhraní `ISignal`. Model se tedy chová jako běžný signál, pouze je pro výpočet hodnot nutné použít sadu parametrů, a to buď dodanou, nebo výchozí pro daný model. Vnitřní logika modelu dovoluje z nadřazeného časového segmentu vybrat referenční signál, který je vždy pro výpočet použit.

8.4.2 Metriky

Metriky dědí od společného předka – třídy `CCommon_Metric`, implementující rozhraní `IMetric`. Toto rozhraní definuje metody pro akumulaci hodnot (resp. rozdílů naměřené a vypočtené hodnoty v daném čase) a pro výpočet hodnoty metriky z takto akumulovaných hodnot.

V rámci výpočtu metrik pro hledání parametrů modelu je výpočet metriky často redukován o vybrané operace, které by měly negativní dopad na výkon. Například pokud je daná metrika definována jako druhá mocnina nějakého výrazu, je odebráno právě toto mocnění – číselně hodnota metriky nikdy není uváděna a při porovnávání dvou stejných metrik se výsledek nezmění.

8.4.3 Výpočet parametrů modelu

Moduly pro výpočet parametrů modelu jsou implementovány v modulu souhrnně označeného jako `stochastic`. Ve své konfiguraci vždy přejímají model, jehož parametry mají počítat, metriku, dle které mají počítat a parametry obou těchto entit. Také přejímají sadu parametrů, které zahrnují do výpočtu jako počáteční odhad.

Navenek poskytují i informaci o stavu výpočtu. Tato informace je vyjádřením průběhu algoritmu, kdy v případě metadiferenciální evoluce se jedná o počet již proběhlých generací vztažený k celkovému počtu generací, a v případě algoritmu NEWUOA jde o počet hotových iterací vztažený na celkový konfigurovaný počet.

Výpočet probíhá vždy synchronně vůči volajícímu, proto filtry spouštějí vždy nové vlákno, ve kterém tato operace probíhá.

8.4.4 Aproximace a interpolace

V současné době jsou aproximační a interpolační metody použity pouze pro dopočítání mezilehlých hodnot naměřených signálů a hodnot derivace. V této práci nebylo požadavkem implementování žádných pokročilejších metod, proto je implementována pouze interpolace lomenou čarou (*spline* křivka 1. řádu).

Každá metoda aproximace nebo interpolace musí implementovat rozhraní `IApproximator` a zpřístupnit právě jednu definovanou metodu pro výpočet hodnot v požadovaných časech.

8.5 Řídicí kód

Pro správné spouštění simulace je nutné mít obalový kód, který inicializuje filtry, příslušné roury, předá konfigurační parametry a celou architekturu spustí.

Celý systém je konfigurovaný souborem ve formátu INI[6]. Tento soubor ob-

[6]formát souboru členěný na sekce a parametry ve formátu klíč-hodnota

sahuje sekce, z nichž každá identifikuje entitu jejím typem, identifikátorem ve formě GUID a pořadím v řetězu entit fall-through architektury. V sekci jsou pak obsaženy parametry, které jsou při inicializaci do entity předány. Příkladem může být následující úryvek konfiguračního souboru:

```
[Filter_006_{14a25f4c-e1b1-85c4-1274-9a0d11e09813}]
Model = {5fd93b14-aaa9-44d7-a8b8-c158318364bd}
Signal = {287b9bb8-b73b-4485-be20-2c8c40983b16}
CalcPastFirstParamSet = true
```

Tento úryvek konfiguruje filtr pro výpočet signálu jako šestý v pořadí. Konfiguruje ho GUID modelu a signálu a dodatečným parametrem, kterým určuje kritérium přepočtu.

Logika řídicího kódu je společná pro mobilní telefon i osobní počítač a je implementována ve třídě `CFilter_Chain_Holder`. Po spuštění je načtena konfigurace, jsou vytvořeny komunikační roury, inicializovány jednotlivé entity architektury a jsou jim předány konfigurační parametry a instance vstupní a výstupní roury.

8.6 Uživatelské rozhraní

Uživatelské rozhraní je jedinou částí systému, která je odlišná pro každou platformu. Užívá v maximální míře společného kódu a odsunutí společné logiky do dynamických knihoven, ale určitá množina funkcionality musela být implementována odděleně.

8.6.1 Osobní počítač

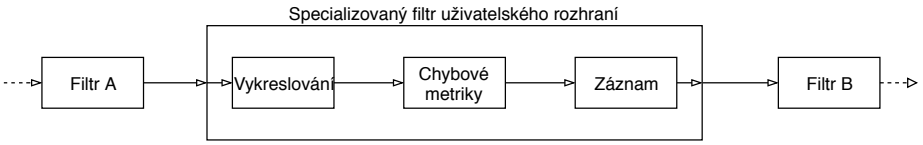
Uživatelské rozhraní aplikace pro osobní počítač bylo implementováno v jazyce C++ užitím knihovny Qt. Implementuje MDI[7] rozhraní. Dále byly implementovány dva hlavní pohledy - *Filters* pro správu konfigurace filtrů v řetězu a *Simulation* pro řízení simulace a vizualizaci.

[7]Multiple documents interface – rozhraní s jedním rodičovským oknem a několika potomky uvnitř

Konfigurace filtrů v řetězu je prováděna v okně *Filters*, ve kterém jsou dva seznamy – jeden s aktuálně používaným řetězem filtrů a jeden s dostupnými filtry, které byly nalezeny v příslušném adresáři na souborovém systému. Po vložení filtru do řetězu je možné otevřít konfigurační okno daného filtru. Toto okno obsahuje veškeré podporované konfigurační volby, které filtr ve svém deskriptoru deklaruje.

Simulační pohled *Simulation* obsahuje ovládací prvky simulace, část obsahující indikátory průběhu výpočtu parametrů modelu a část vizualizační, která zabírá většinu okna.

Aplikace pro osobní počítač dále definuje dva specializované filtry, které zapojuje do řetězu. Jedním z nich je filtr uživatelského vstupu, kterým lze do řetězu z jeho začátku odesílat vybrané zprávy – zde zejména kvůli ovládání simulace. Druhým filtrem je filtr grafických a jiných výstupů. Ten obaluje několik filtrů, konkrétně se jedná o filtr vykreslovací, filtr pro výpočet chyb a filtr, který provádí záznam. Schéma lze vidět na obrázku 8.2.



Obrázek 8.2: Schéma části filtrového řetězu, ve kterém figuruje i specializovaný filtr uživatelského rozhraní a jím obalené filtry

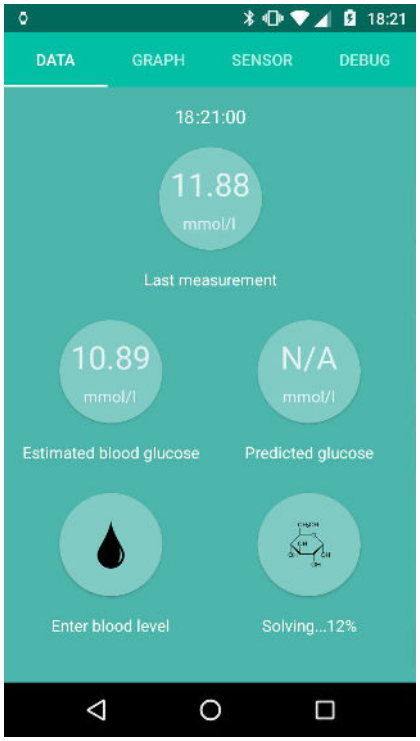
8.6.2 Mobilní telefon

Uživatelské rozhraní mobilní aplikace bylo implementováno v jazycích C# a XAML v rámci technologie Xamarin a knihovny Xamarin.Forms.

Aplikace je členěna na tři pohledy – první obsahuje nejnovější načtená a vypočtená data, druhá grafy s vývojem hodnot a denním přehledem a třetí propojení se senzorem. Po prvním spuštění uživatel musí ve třetí záložce vyhledat senzor a propojit ho s aplikací. Ta si senzor zapamatuje a při dalším spuštění ho sama vyhledá a pokusí se k němu připojit. V současné době podporuje aplikace pouze senzory implementující standard IEEE 11073 (Bluetooth Low-Energy), díky modularitě řešení je však možné přidat další odpovídající implementací třídy `DeviceAdapterBase`.

První záložka obsahuje měřenou hodnotu ze senzoru (koncentrace glukózy

v intersticiální tekutině), vypočtenou hodnotu (koncentrace glukózy v krvi) a datum a čas měření. Také je pro budoucí úpravy připraveno pole pro předpovězenou koncentraci glukózy. Snímek obrazovky s první záložkou lze vidět na obrázku 8.3. Druhá záložka obsahuje grafy, které jsou generovány modulem implementovaným v rámci architektury. Ten je pouze parametrizován rozměry takovými, aby se grafy vešly na obrazovku mobilního telefonu a měly odpovídající poměr stran.



Obrázek 8.3: Snímek obrazovky mobilního telefonu se spuštěnou aplikací

Aplikace dále umožňuje v první záložce zadat hodnotu kalibrace, tedy hodnotu koncentrace glukózy v krvi získanou měřením externím přístrojem (glukometrem). Ta je odeslána do senzoru jako kalibrační a je také odeslána do architektury, aby ji mohl výpočetní model zařadit jako referenční pro výpočet.

Pro potřeby aplikace je dále implementována dvojice filtrů – filtr pro přejímání hodnot ze senzoru a z uživatelského vstupu a filtr pro „export“ zpráv do kódu rozhraní (odtud je brána vypočtená hodnota koncentrace). Tyto filtry jsou implementovány třídami `CDevice_Filter`, respektive `CInterop_Export_Filter`.

Konfigurace architektury je prováděna pouze staticky v čase kompilace a to

z toho důvodu, že běžný uživatel nemusí mít přístup k nastavení výpočetních částí systému. Výstupy však konfigurovatelné jsou – například konfigurace jednotek, ve kterých je zobrazen výstup ($mmol/l$ nebo mg/dl), možnosti notifikace a jiné.

8.7 „Chytré“ hodinky

Do chytrých hodinek byla implementována podpora zobrazování datových proudů v rámci systému *complications*. Z mobilní aplikace jsou poskytnuty 2 datové proudy – jeden s naměřenou koncentrací ze senzoru a jeden s vypočtenou koncentrací v krvi. Systém dále nepředepisuje uživateli, který datový proud má nebo nemá mít zobrazen, případně kam je umístěn.

Na obrázku 8.4 lze vidět snímek obrazovky chytrých hodinek s volně dostupnou *watch face*[8] s názvem *HUD*[9], která ve dvou pozicích pro datové proudy zobrazuje oba poskytnuté údaje.

Obrázek 8.4: Snímek obrazovky chytrých hodinek s prezentací datových proudů systémem *complications*

Tato část byla rovněž implementována užitím technologie Xamarin. Komunikace s mobilním telefonem je zajištěna instancí třídy `GoogleApiClient` a modulů `DataApi`, které jsou poskytovány přímo vývojovou sadou systému Android. Aktualizace probíhá při každé změně hodnoty datového proudu, a to je pro reálný senzor vždy v řádu jednotek minut.

[8]hlavní pohled chytrých hodinek s ukazatelem času a dalších údajů
[9]dostupné v balíku *ustwo Watch Faces* v obchodu Google Play

8.8 Simulátor reálného zařízení

Jako vývojová platforma byl vybrán modul Texas Instruments LAUNCHXL-CC2640R2, jelikož obsahuje potřebné hardwarové vybavení (zejména pak modul Bluetooth Low-Energy) a byl dostupný k zapůjčení na katedře.

Základ aplikace využívá předpřipravený rámec od autorů implementace Bluetooth Low-Energy zásobníku, který je připraven tak, aby programátor mohl pouze definovat a implementovat jednotlivé profily (jejich charakteristiky a další). Je vystavěn na operačním systému TI-RTOS. To je operační systém reálného času, který implementuje primitiva pro realizaci multitaskingu, synchronizaci, komunikaci s periferiemi a další.

Byl definován a implementován profil pro kontinuální měření glukózy standardizovaný v rámci IEEE 11073. Základní implementovanou podmnožinou je získávání hodnot měření buď bez explicitního vyžádání (tedy normální režim kontinuálního měření), nebo s vyžádáním skrze charakteristiku *CGM Specific Control Ops* (získávání hodnot z historie).

Do modulu jsou nahrána testovací data pouze v čase překladu. To je pro potřeby této práce dostačující – lze ověřit správnost implementace mobilní aplikace. Simulátor pak periodicky tyto hodnoty vybírá a vkládá je do „databáze“ (v podobě cyklické fronty) s patřičným označením časovou značkou.

9 Dosažené výsledky

Aplikace byla testována s anonymizovanými daty z reálného měření, která byla poskytnuta Diabetologickým centrem Fakultní nemocnice Plzeň[1]. Data však nejsou použitelná pro porovnání kvality modelů – byla dodána data různorodá z různých typů senzorů a časových úseků, a to zejména kvůli možnosti implementovat a ověřit implementaci na různých vstupních formátech. Tato kapitola se věnuje dosaženým výsledkům, měření spotřeby elektrické energie na mobilním zařízení a dalším nárokům na prostředky systému.

Tato práce zapadá do širšího projektu, předchozí experimenty a měření na jiných zařízeních jsou obsaženy v [22].

9.1 Scénář testování

Testování proběhlo zejména na mobilní verzi aplikace. Desktopová aplikace byla také testována, a to na správnost dosažených výsledků a kvalitu výstupů.

Pro mobilní aplikaci bylo měřeno jednak množství procesorového času vynaloženého na výpočet parametrů metod s danými parametry, a jednak spotřeba elektrické energie. V textu je dále algoritmus metadiferenciální evoluce označován zkratkou *MetaDE*. Pro všechny testy byla použita metrika Crosswalk, jakožto doporučená metrika pro výpočty s daty daného charakteru (uvažuje časové uspořádání hodnot). Pro MetaDE byla použita populace velikosti 100 a počet generací byl nastaven na 1000. Aplikace pro osobní počítač pak používala populaci o velikosti 100 a počet generací byl nastaven na 10000.

Celkem proběhla čtyři měření:

1. MetaDE, difúzní model
2. MetaDE, model Steil-Rebrinové
3. NEWUOA, difúzní model

[1]Diabetologické centrum FN Plzeň, Alej Svobody 80, 323 00 Plzeň

4. NEWUOA, model Steil-Rebrinové

Měření probíhalo na mobilním telefonu Motorola MotoG (3rd gen) s operačním systémem Android ve verzi 6.0.1 bez dodatečných úprav. Knihovny byly zkompilovány kompilátorem `clang` verze 5.0 pro ABI `armeabi-v7a` (32-bitové) s použitím instrukční sady NEON a nejvyšší dostupnou úrovní optimalizací (přepínač `-O3`).

Pro potřeby měření byla aplikace dočasně upravena, aby do podlehlé architektury odeslala jeden celý naměřený segment a vynutila přepočet parametrů modelu (tedy spustila příslušný algoritmus). Poté aplikace vyčkala, až algoritmus skončil, poskytl nové parametry modelu, a ukončila se. Zároveň bylo vypnuto vykreslování, modul Bluetooth a další moduly, které nebyly předmětem měření a zkreslovaly by výsledky.

9.2 Měření spotřeby

Spotřeba elektrické energie a množství procesorového času spotřebovaného mobilní aplikací bylo měřeno nástrojem `adb`, který komunikuje s modulem operačního systému Android pro shromažďování statistik o používání zařízení[29]. Nástroj `adb` je nástrojem komplexním, pro komunikaci s tímto modulem postačí podprogram `dumpsys` s parametrem `batterystats`. Před každým měřením byl telefon připojen do počítače USB kabelem a tyto statistiky byly resetovány příkazem

```
adb shell dumpsys batterystats --reset
```

Poté byl telefon odpojen, aplikace spuštěna a bylo vyčkáno, než byla díky dočasné modifikaci ukončena (běh algoritmu skončil a byly vypočteny parametry). Následně byl telefon připojen a byl spuštěn příkaz

```
adb shell dumpsys batterystats
```

Ten poskytnul komplexní hlášení o všech aplikacích, jejich spotřebách a další informace, které nejsou pro toto měření relevantní. Ve výstupu byl dohledán záznam aplikace a informace o procesorovém času a spotřebě elektrické energie, respektive spotřebované kapacitě baterie. Poté byly statistiky opět

resetovány a celé měření bylo opakováno. Celkem tak proběhlo 10 měření, a to pro každou ze zmíněných kombinací algoritmu a modelu. Také proběhlo měření bez spouštění algoritmu, aby byl změřen základ, který je potřeba pro spuštění aplikace a inicializaci architektury.

Z naměřených hodnot byl vypočítán průměr a směrodatná odchylka. Také bylo nástrojem hlášeno aktuální napětí $U = 4.134V$.

Algoritmus, model	E_t [s]	σ_t [s]	E_p [mAh]	σ_p [mAh]
žádný	5.767	0.2597	0.0838	0.0163
MetaDE, difúzní	83.102	0.5864	1.8560	0.0583
MetaDE, Steil-Rebrin	81.049	0.6804	1.8080	0.0535
NEWUOA, difúzní	6.051	0.4247	0.1142	0.0263
NEWUOA, Steil-Rebrin	6.544	0.1590	0.1174	0.0369

Tabulka 9.1: Průměry procesorového času (E_t) a spotřeby el. náboje (E_p) s odpovídajícími směrodatnými odchylkami (σ_t, resp. σ_p)

Počítejme nyní pouze s průměrnými časy a spotřebami. Od časů naměřených pro jednotlivé metody odečtěme základ v podobě průměrů pro měření bez algoritmu. Tím získáme průměrné časy a spotřeby jednotlivých algoritmů a modelů bez režie v podobě spouštění, inicializace a ukončování aplikace.

Spotřebovaná energie (resp. její střední hodnota) byla vypočtena vztahem:

$$E_e = E_Q \times \frac{U}{1000} \tag{9.1}$$

Algoritmus, model	E_t [s]	E_Q [mAh]	E_e [Wh]
MetaDE, difúzní	77.335	1.7722	0.0073263
MetaDE, Steil-Rebrin	75.282	1.7242	0.0071278
NEWUOA, difúzní	0.284	0.0304	0.0001257
NEWUOA, Steil-Rebrin	0.777	0.0336	0.0001389

Tabulka 9.2: Průměry procesorového času (E_t), spotřeby el. náboje (E_Q) a energie (E_e) po odečtení průměrných časů a spotřeb naměřených z běhu bez výpočtu

V tabulce 9.2 lze vidět velký rozdíl mezi oběma použitými algoritmy. To lze odůvodnit několika fakty. Metadiferenciální evoluce je algoritmus nedeterministický s velkým počtem iterací (generací; zde 1000), kdy v každé generaci je nutné přepočítat metriku pro každého člena populace (zde 100), kdy může

být použita různá strategie pro mezigenerační křížení a mutace, což také přidá další zátěž. Oproti tomu NEWUOA je algoritmus s deterministickým průběhem, malým počtem iterací a nepracuje s žádnou analogií populace – pouze aproximuje dílčí části kvadratickou formou, což je podstatně rychlejší metoda optimalizace, než stochastický algoritmus, kterým je právě MetaDE.

Také lze pozorovat to, že u MetaDE trvá delší dobu výpočet parametrů modelu difúzního, než modelu Steil-Rebrinové, ale u algoritmu NEWUOA je tomu naopak. Tato skutečnost se dá vysvětlit tím, že difúzní model má více parametrů (6), než model Steil-Rebrinové (4), a tedy je potřeba více operací pro výpočet. Model Steil-Rebrinové proti modelu difúznímu ale pracuje s derivacemi (resp. numerickými derivacemi) referenčního signálu. Tyto rozdíly se pak mohou v různě dlouhých časových úsecích projevovat jinak.

9.3 Funkčnost

Obě aplikace byly testovány na funkčnost použitím dodaných dat. Tato data byla dodána za účelem možnosti implementovat různé formáty pro zpracování aplikací, nelze je tedy použít pro porovnání kvality modelů.

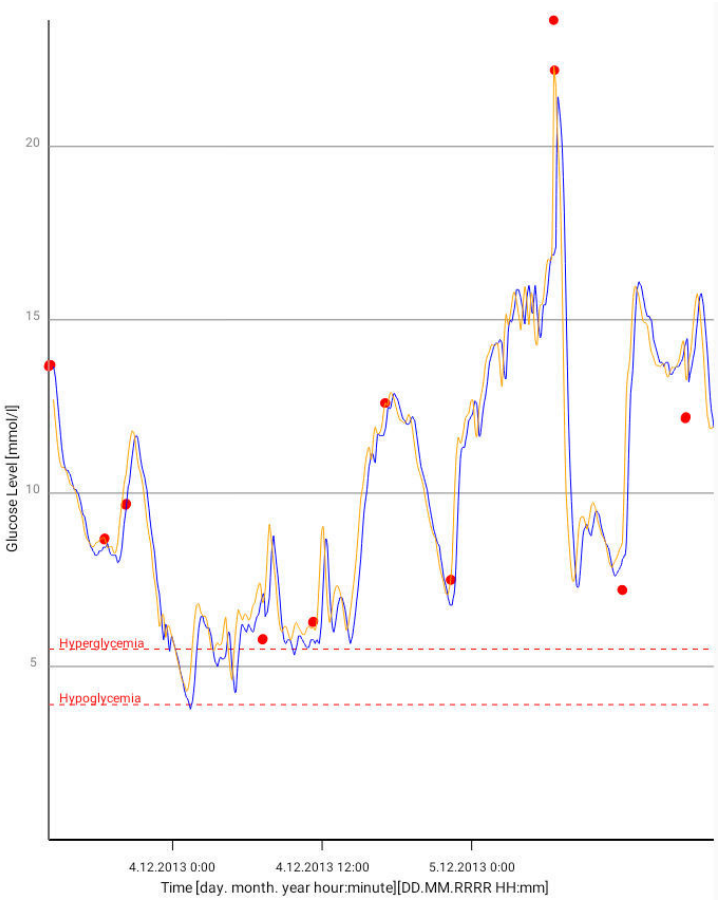
	MetaDE		NEWUOA	
	PC	Mobil	PC	Mobil
p	1.053	1.056	1.091	1.091
cg	-0.011	-0.011	-0.016	-0.016
c	-0.681	-0.714	-0.174	-0.174
dt	0.012	0.012	0.023	0.023
k	0	0	<0.001	<0.001
h	0	0	0.019	0.019

Tabulka 9.3: Parametry difúzního modelu vypočtené oběma algoritmy v každém z prostředí; zaokrouhleno na 3 desetinná místa

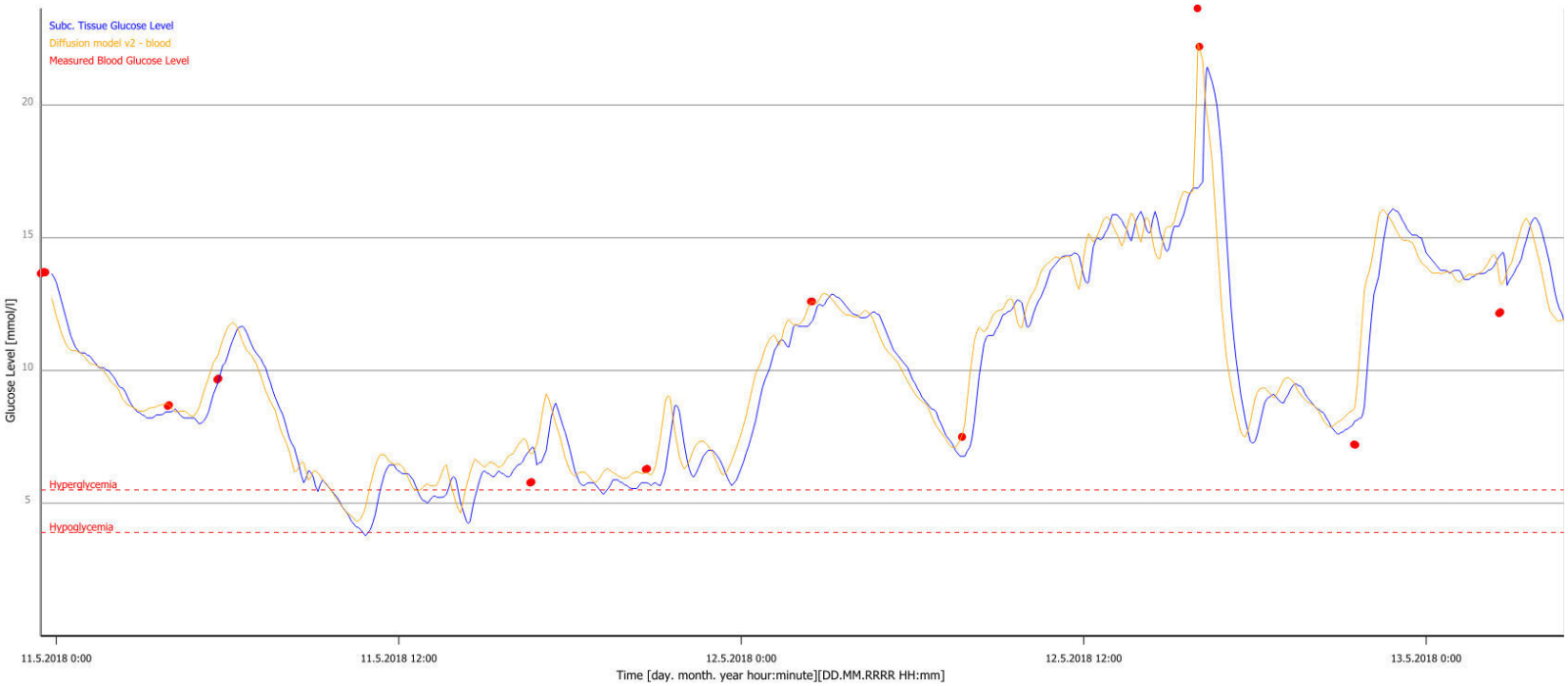
V tabulkách 9.3 a 9.4 lze vidět výsledky výpočtu v každém z prostředí oběma algoritmy. Parametry se mezi prostředími příliš neliší, což lze za předpokladu, že je výpočetní metoda zatížena nedeterminismem (a tedy nelze docílit identických hodnot, pouze blízkých), označit za správný výsledek. Obrázky 9.1 a 9.2 pak ukazují vykreslený graf v obou aplikacích. V grafech je modrou čarou znázorněna koncentrace glukózy v podkoží a žlutou čarou vypočtená koncentrace v krvi.

	MetaDE		NEWUOA	
	PC	Mobil	PC	Mobil
τ	-0.001	0.006	-0.001	-0.001
α	1.112	0.979	0.978	0.979
β	-1.159	0	-0.038	-0.039
γ	115.230	-12.811	0.045	0.044

Tabulka 9.4: Parametry modelu Steil-Rebrinové vypočítané oběma algoritmy v každém z prostředí; zaokrouhleno na 3 desetinná místa



Obrázek 9.1: Graf vykreslený v aplikaci pro mobilní telefon pro testovaný měřený segment; parametry byly spočítány ze všech dostupných hodnot segmentu algoritmem metadiferenciální evoluce s metrikou Crosswalk



Obrázek 9.2: Graf vykreslený v aplikaci pro osobní počítač pro testovaný měřený segment; parametry byly spočítány ze všech dostupných hodnot segmentu algoritmem metadiferenciální evoluce s metrikou Crosswalk

Na tomto segmentu byla provedena analýza chyb užitím standardních statistických metrik. Výsledky lze vidět v tabulkách 9.5 a 9.6. Lze vidět, že parametry vykazují stejnou a v některých případech i lepší kvalitu, než z aplikace na osobním počítači (např. u difúzního modelu a algoritmu NEWUOA byly na mobilu nalezeny parametry s menší minimální chybou).

Tabulky obsahují relativní chyby vypočtené vztahem:

$$\delta_r = (\frac{x_0}{x} - 1) \times 100[\%] \tag{9.2}$$

	MetaDE		NEWUOA	
	PC	Mobil	PC	Mobil
Průměr	5.0%	5.0%	16.4%	16.5%
Směrodatná odchylka	6.6%	6.7%	13.7%	14.1%
Minimum	0.0%	0.0%	2.6%	2.5%
1. kvartil	0.1%	0.2%	11.6%	12.1%
Medián	1.5%	1.5%	13.1%	13.5%
3. kvartil	7.9%	7.9%	16.9%	16.9%
95. percentil	18.9%	19.0%	56.4%	58.2%
Maximum	21.1%	21.1%	65.0%	65.1%

Tabulka 9.5: Chybové metriky (relativní chyby) relevantní pro parametry difúzního modelu vypočítané oběma algoritmy v každém z prostředí; hodnoty jsou uvedeny v procentech, zaokrouhleny na 1 desetinné místo, byly počítány z nezaokrouhlených parametrů

	MetaDE		NEWUOA	
	PC	Mobil	PC	Mobil
Průměr	8.2%	10.3%	9.6%	9.7%
Směrodatná odchylka	10.5%	9.6%	8.6%	8.8%
Minimum	0.0%	0.0%	0.0%	0.1%
1. kvartil	0.3%	3.6%	4.4%	4.5%
Medián	4.0%	5.2%	5.7%	6.1%
3. kvartil	14.7%	20.0%	16.7%	18.8%
95. percentil	29.2%	28.8%	25.4%	27.5%
Maximum	31.1%	29.2%	26.9%	28.9%

Tabulka 9.6: Chybové metriky relevantní pro parametry modelu Steil-Rebrinové vypočítané oběma algoritmy v každém z prostředí

10 Závěr

Zadání bylo splněno v celém rozsahu. V kapitole 2 byla stručně popsána dynamika glukózy v lidském těle a nemoc diabetes mellitus. Kapitola 3 se pak věnovala modelování dynamiky glukózy v lidském těle, algoritmům pro hledání parametrů těchto modelů a příslušných metrik. Dále byly v kapitole 4 analyzovány způsoby měření koncentrace glukózy v krvi a přenos navzorkovaného signálu do dalšího zařízení, na kterou navazovala kapitola 5, která obsahovala analýzu metod zpracování a prezentace dat. Následovala kapitola 6, která se věnovala simulačním architekturám a kapitola 7, která se věnovala návrhu systému založeného na *fall-through* architektuře. Celé řešení bylo implementováno, což popisuje kapitola 8. Nakonec se kapitola 9 věnuje testování a měření hotových aplikací.

Z praktické části byl implementován aplikační software pro mobilní telefon schopný komunikovat se senzorem pro kontinuální měření koncentrace glukózy v podkoží. Také byl vyvinut simulátor senzoru implementující podmnožinu standardu IEEE 11073 se specializací na kontinuální měření koncentrace glukózy. Dále byl implementován aplikační software pro osobní počítač, který umí spouštět a řídit simulaci a vizualizovat výsledky výpočtů v grafech a chybových mřížkách.

Výsledek práce je prototypem CGMS systému nové generace pro monitoraci koncentrace glukózy na straně pacienta, který vlastní mobilní telefon s operačním systémem Android nebo iOS, a jednak pro výzkumné účely v oblasti modelování dynamiky glukózy.

Hotové aplikace byly s dodanými daty otestovány na mobilním telefonu a osobním počítači. Byla ověřena funkčnost obou aplikací a také byla změřena spotřeba elektrické energie pro vykonání algoritmu pro výpočet parametrů modelu.

Vzhledem k použité architektuře je možné systém pohodlně rozšiřovat přidáním dalšího modulu, který implementuje popsané rozhraní. To je klíčovou vlastností celého řešení, jelikož modely, metriky a další části systému se neustále vyvíjí v rámci výzkumu na Katedře informatiky a výpočetní techniky i mimo ni.

Dalším možným rozšířením je však i napojení na simulátor lidského těla, respektive mechanismů homeostázy glukózy. To by mohlo přinést lepší prostředí pro vývoj modelů dynamiky glukózy, vhodné metriky nebo dalších příbuzných algoritmů. K simulátoru lidského těla může být připojena i simulovaná inzulinová pumpa a simulace příjmu potravy, aby bylo možné modely dále zpřesňovat.

Literatura

[1] Health informatics–Personal health device communication - Part 10425: Device Specialization–Continuous Glucose Monitor (CGM). *IEEE Std 11073-10425-2017 (Revision of IEEE Std 11073-10425-2014)*. Leden 2018, s. 1–83. doi: 10.1109/IEEESTD.2018.8272357.

[2] ISO/IEC/IEEE International Standard - Health Informatics – Personal Health Device Communication - Part 20601: Application Profile - Optimized Exchange Protocol. *ISO/IEEE 11073-20601:2016(E)*. Červen 2016. doi: 10.1109/IEEESTD.2016.7842820.

[3] Acciaroli, M. G. a. V. – Facchinetti, A. – Sparacino, G. Calibration of Minimally Invasive Continuous Glucose Monitoring Sensors: State-of-The-Art and Current Perspectives. *MDPI Biosensors*. 2018, 8, 1, s. 24. doi: 10.3390/bios8010024.

[4] Ahlqvist, E. – others. Novel subgroups of adult-onset diabetes and their association with outcomes: a data-driven cluster analysis of six variables. *The Lancet. Diabetes and endocrinology*. 2018. ISSN 2213-8595. doi: 10.1016/S2213-8587(18)30051-2.

[5] *Bluetooth Low Energy Protocol Stack* [online]. Texas Instruments, 2013. [cit. 11.4.2018]. Dostupné z: `http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_00_00_22/docs/blestack/html/ble-stack/index.html`.

[6] Bronzino, J. *Medical Devices and Systems*. The Biomedical Engineering Handbook, Fourth Edition. CRC Press, 2006. Dostupné z: `https://books.google.cz/books?id=0Qj0-iecCkQC`. ISBN 9781420003864.

[7] Cengiz, E. – Tamborlane, W. V. A Tale of Two Compartments: Interstitial Versus Blood Glucose Monitoring. *Diabetes Technology and Therapeutics*. 2009, 11, 1, s. 11–16. doi: 10.1089/dia.2009.0002.

[8] Dalvi, N. *Glucose Meter Reference Design*. Microchip Technology Inc., 2013. Dokument číslo: 00001560A.

[9] *Data o diabetu v ČR* [online]. Diabetická asociace ČR, 2015. [cit. 7.4.2018]. Dostupné z: `http://www.diabetickaasociace.cz/co-je-diabetes/data-o-diabetu-v-cr/`.

[10] Favero, S. D. et al. Improving Accuracy and Precision of Glucose Sensor Profiles: Retrospective Fitting by Constrained Deconvolution. *IEEE Transactions on Biomedical Engineering*. April 2014, 61, 4, s. 1044–1053. ISSN 0018-9294. doi: 10.1109/TBME.2013.2293531.

[11] *Global report on diabetes* [online]. World Health Organisation, 2016. [cit. 7.4.2018]. Dostupné z: `http://apps.who.int/iris/bitstream/10665/204871/1/9789241565257_eng.pdf?ua=1`.

[12] Güemes, M. – Rahman, S. A. – Hussain, K. What is a normal blood glucose? *Archives of Disease in Childhood*. 2016, 101, 6, s. 569–574. doi: 10.1136/archdischild-2015-308336.

[13] Hall, J. *Guyton and Hall Textbook of Medical Physiology E-Book*. Guyton Physiology. Elsevier Health Sciences, 2015. Dostupné z: `https://books.google.cz/books?id=krLSCQAAQBAJ`. ISBN 9780323389303.

[14] Hovorka, R. et al. Nonlinear model predictive control of glucose concentration in subjects with type 1 diabetes. *Physiological Measurement*. 2004, 25, 4, s. 905. Dostupné z: `http://stacks.iop.org/0967-3334/25/i=4/a=010`.

[15] Hughes, R. et al. *Continuous Glucose Monitoring Service* [online]. Bluetooth SIG, 2014. [cit. 16.4.2018]. Verze 1.0.1. Dostupné z: `https://www.bluetooth.com/specifications/gatt`.

[16] *IDF Diabetes Atlas* [online]. International Diabetes Federation, 2017. [cit. 7.4.2018]. Dostupné z: `http://www.idf.org/diabetesatlas`.

[17] Kamath, S. – Lindh, J. *Measuring Bluetooth Low Energy Power Consumption* [online]. Texas Instruments, 2012. [cit. 16.4.2018]. Dokument číslo: SWRA347a.

[18] Klueh, U. et al. Metabolic Biofouling of Glucose Sensors in Vivo: Role of Tissue Microhemorrhages. *Journal of Diabetes Science and Technology*. 2011, 5, 3, s. 583–595. doi: 10.1177/193229681100500313.

[19] Koutny, T. Crosswalk – a time-ordered metric. In *EMBEC & NBC 2017*, s. 884–887, Singapore, 2018. Springer Singapore. ISBN 978-981-10-5122-7.

[20] Koutny, T. Blood glucose level reconstruction as a function of transcapillary glucose transport. *Computers in Biology and Medicine*. 2014, 53, s. 171 – 178. doi: https://doi.org/10.1016/j.compbiomed.2014.07.017.

[21] Koutny, T. Using meta-differential evolution to enhance a calculation of a continuous blood glucose level. *Computer Methods and Programs in Biomedicine*. 2016, 133, s. 45 – 54. doi: https://doi.org/10.1016/j.cmpb.2016.05.011.

[22] Koutny, T. – Siroky, D. Analyzing Energy Requirements of Meta-Differential Evolution for Future Wearable Medical Devices. *World Congress on Medical Physics and Biomedical Engineering, Prague*. 2018.

[23] Leach, P. et al. A Universally Unique IDentifier (UUID) URN Namespace. RFC 4122, IETF, July 2005. Dostupné z: `https://tools.ietf.org/rfc/rfc4122.txt`.

[24] Ma, J. et al. Research on Online Measurement Method of Smartphone Energy Consumption. In *2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, s. 443–447, Nov 2014. doi: 10.1109/3PGCIC.2014.90.

[25] Omre, A. H. Bluetooth Low Energy: Wireless Connectivity for Medical Monitoring. *Journal of Diabetes Science and Technology*. 2010, 4, 2, s. 457–463. doi: 10.1177/193229681000400227.

[26] Padgette, J. – Scarfone, K. – Chen, L. *Guide to Bluetooth Security* [online]. National Institute of Standards and Technology, 2017. [cit. 16.4.2018]. Dokument číslo: 800-121 rev. 1.

[27] Pappada, S. M. – Cameron, B. D. – Rosman, P. M. Development of a Neural Network for Prediction of Glucose Concentration in Type 1 Diabetes Patients. *Journal of Diabetes Science and Technology*. 2008, 2, 5, s. 792–801. doi: 10.1177/193229680800200507.

[28] Powell, M. J. D. *The NEWUOA software for unconstrained optimization without derivatives*, s. 255–297. Springer US, Boston, MA, 2006. doi: 10.1007/0-387-30065-1_16.

[29] *Profile Battery Usage with Batterystats and Battery Historian* [online]. Google, Inc., 2018. [cit. 8.5.2018]. Dostupné z: `https://developer.android.com/studio/profile/battery-historian`.

[30] *Radio Frequency Wireless Technology in Medical Devices* [online]. Food and Drug Administration, U.S. Department of Health and Human Services, 2013. [cit. 11.4.2018]. Dokument číslo: ucm077272.

[31] Rebrin, K. – Steil, G. M. Can Interstitial Glucose Assessment Replace Blood Glucose Measurements? *Diabetes Technology & Therapeutics*. 2000, 2, 3, s. 461–472. doi: 10.1089/15209150050194332. PMID: 11467349.

[32] Schricker, B. C. – Lippe, S. R. Using the high level architecture to implement selective-fidelity. In *37th Annual Simulation Symposium, 2004. Proceedings.*, s. 246–253, April 2004. doi: 10.1109/SIMSYM.2004.1299489.

[33] Shendurse, A. – Khedkar, C. Glucose: Properties and Analysis. In Caballero, B. – Finglas, P. M. – Toldrá, F. (Ed.) *Encyclopedia of Food and Health*. Oxford: Academic Press, 2016. s. 239 – 247. doi: https://doi.org/10.1016/B978-0-12-384947-2.00353-6.

[34] Statista. *Number of smartphone users worldwide 2014-2020* [online]. statista.com, 2018. [cit. 30.4.2018]. Dostupné z: `https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/`.

[35] Topçu, O. – Oğuztüzün, H. Guide to Distributed Simulation with HLA. In *Simulation Foundations, Methods and Applications*, s. 1–307. Springer International Publishing, 2017. doi: 10.1007/978-3-319-61267-6.

[36] Vaddiraju, S. et al. Technologies for Continuous Glucose Monitoring: Current Problems and Future Promises. *Journal of Diabetes Science and Technology*. 2010, 4, 6, s. 1540–1562. doi: 10.1177/193229681000400632.

A Programátorská dokumentace

Tato příloha obsahuje detailnější popis rozhraní tříd a knihoven, vztahy jednotlivých objektů v architektuře a vybrané datové typy, které byly specifikovány pro přenos informací mezi moduly a knihovnami.

Konvence

Projekt respektuje jednotné konvence, z nejdůležitějších lze uvést:

- základní odsazování tabulátorem
- identifikátory tříd, struktur a typů vždy začínají vyhrazeným písmenem, zbytek identifikátoru respektuje *snake_case* s velkými počátečními písmeny
  - `I` pro rozhraní (resp. abstraktní třídy)
  - `C` pro třídy implementace (bez čistě virtuálních metod)
  - `T` pro typový alias vytvořený užitím `using` (popř. `typedef`)
  - `S` pro typový alias sdíleného ukazatele
  - `N` pro výčtové typy (`enum` a `enum class`)
- identifikátory atributů tříd začínají vždy malým písmenem `m`, pokračují písmenem velkým
- každý hlavičkový soubor začíná definicí `#pragma once` (tedy se nevyužívá makro)
- každá metoda, kterou lze volat přes rozhraní dynamické knihovny má návratový typ `HRESULT` a používá makro s definicí volací konvence `IfaceCalling`
- všechny třídy, které mohou svou instanci exportovat do jiných knihoven, implementují rozhraní `refcnt::IReferenced`, volitelně hotovou implementací `refcnt::CReferenced`

- nikdy není přímo voláno `AddRef` nebo `Release` nad objektem s vlastním čítáním referencí – vždy je využito příslušných továrních metod, v generických implementacích funkce `Manufacture_Object` a vždy je pro lokální uložení objekt převeden na instanci sdíleného ukazatele metodou `refcnt::make_shared_reference` (popř. explicitně typovaná varianta `refcnt::make_shared_reference_ext`)
- nikdy není explicitně použita dynamická (de)alokace operátory `new` a `delete` (popř. funkcemi `malloc()`, `free()` a obdobami), vždy je využito dostupných prostředků knihovny nebo jazyka C++
- pro přenos struktur dynamických velikostí (pole, řetězce, aj.) rozhraním dynamických knihoven jsou použity příslušné specializace šablony kontejneru `refcnt::IVector_Container`, popř. typ `refcnt::wstr_container`, `refcnt::str_container` nebo `refcnt::double_container`

Fall-through architektura

Výkonná část řešení je dělena na tři části – společnou část (`common`), specifickou (`core`) a část pro nízkopříkonová zařízení (`low-power`), jako je třeba mobilní telefon. Tyto části jsou dále na souborovém systému členěny na komponenty nebo logicky oddělené celky:

- `common` – sdílená část všech modulů a aplikací systému
  - `desktop-console` – sdílený kód uživ. rozhraní
  - `iface` – definice rozhraní
  - `lang` – překladové konstanty a sdílené řetězcové identifikátory
  - `resource` – sdílené metainformace pro sestavení na OS Windows
  - `rtl` – implementace sdílených entit
- `core` – knihovny s jednotlivými moduly (dynamickými knihovnami)
  - `approx` – aproximace a interpolace
  - `consume` – filtr pro ukončení lineární architektury a uvolnění prostředků
  - `data` – filtr pro načítání z databáze a ze souboru
  - `drawing` – filtr pro vykreslování

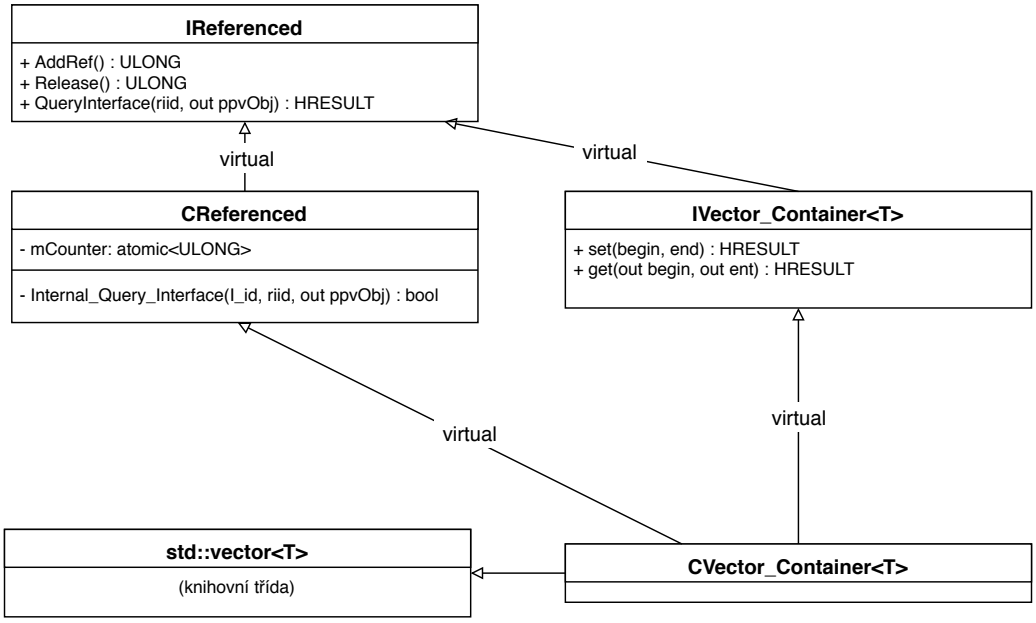
  - `factory` – tovární knihovna pro vyhledávání a vytváření jednotlivých modulů
  - `log` – filtr pro tvorbu záznamu ze simulace nebo průběhu měření
  - `metric` – metriky pro výpočty parametrů modelu a filtr pro výpočet chybových metrik pro uživ. rozhraní
  - `model` – modely dynamiky glukózy
  - `network` – filtr pro přenos zpráv architektury po síti
  - `signal` – naměřené signály a filtr pro výpočet signálů na základě parametrů modelu
  - `solver` – filtr pro výpočet parametrů modelu
  - `stochastic` – algoritmy pro výpočet parametrů modelu
- `low-power`
  - `device` – filtr pro vstup dat z vnějšího kódu
  - `interop-export` – export dat z architektury do vnějšího kódu

Podpůrné třídy

Pro celé řešení je společné několik rozhraní a tříd, které plní obecnou funkci. Každá taková entita má definované rozhraní, které je odděleno v hlavičkovém souboru v podsložce `iface` části `common`.

Na obrázku A.1 lze vidět třídy a rozhraní pro čítání odkazů v meziknihovních ukazatelích. Všechny tyto definice jsou obsaženy ve jmenném prostoru `refcnt`. Vždy je pro vytváření objektů implementující rozhraní `IReferenced` (popř. implementaci `CReferenced`) využito tovární funkce `Manufacture_Object`. Lze si všimnout, že dědičnost je definována jako virtuální – to z důvodu „diamantové“ vícenásobné dědičnosti a zjednoznačnění vazeb implementovaných metod. Také je vidět definice kontejnerové struktury `IVector_Container`, resp. `CVector_Container`, která vnitřně využívá knihovní implementaci `std::vector`, ale poskytuje rozhraní pro možnost meziknihovní interakce. Tato instance nikdy není vytvářena přímo, je vždy využito funkce `Create_Container` nebo libovolné jiné, která tuto funkci volá.

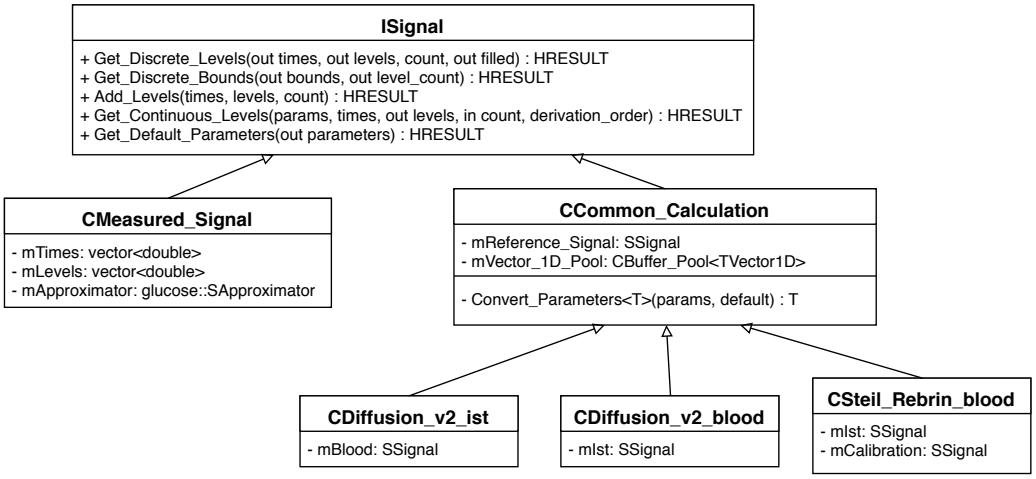
Na diagramu A.2 je vidět UML diagram tříd signálů. Tyto signály mají společného předka, respektive implementují stejné rozhraní `ISignal`. Ten



Obrázek A.1: UML diagram tříd pro rozhraní a implementaci meziknihovního čítání odkazů

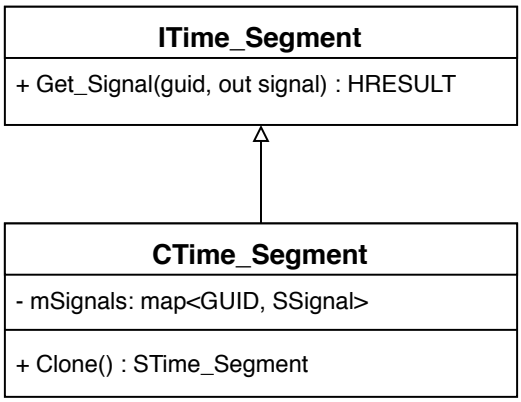
definuje metody pro přístup k hodnotám, které se mohou lišit pro naměřený a vypočtený signál:

- `Get_Discrete_Levels` – získá uložené hodnoty a naplní jimi parametry `times` a `levels`, nejvýše však počet `count`. Finální uložený počet je naplněn do `filled`
- `Get_Discrete_Bounds` – naplní parametr `bounds` okrajovými hodnotami (minima a maxima) časové osy a osy koncentrace, a parametr `level_count` počtem hodnot
- `Add_Levels` – přidá nové hodnoty z parametrů `times` a `levels`, tato pole musí mít stejnou velikost dodanou v parametru `count`; platné pouze pro naměřený signál
- `Get_Continuous_Levels` – s dodanými parametry modelu `params` a časy `times` naplní pole `levels` příslušnými hodnotami (aproximace u naměřeného signálu), počtem nejvýše `count`. Lze získat hodnoty signálu nebo jeho derivace, to řídí parametr `derivation_order`
- `Get_Default_Parameters` – pro signály modelu vrací výchozí parametry; platné pouze pro vypočtený signál



Obrázek A.2: UML diagram tříd pro rozhraní a implementaci signálů

Třída naměřeného signálu ve své instanci ukládá časy a naměřené hodnoty a také instanci modulu pro aproximaci a interpolaci. Každý vypočtený signál v sobě uchovává instanci referenčního signálu, a navíc může každý ukládat ještě dodatečné signály, které k výpočtu používá.

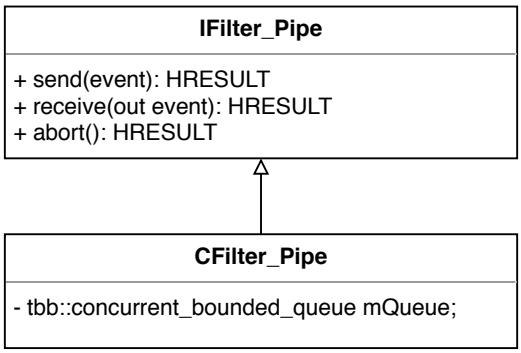


Obrázek A.3: UML diagram tříd pro rozhraní a implementaci časových segmentů

Na diagramu A.3 lze vidět diagram tříd segmentu, který slučuje všechny signály z měření.

- `Get_Signal` – získá uložený signál se zadaným identifikátorem `guid` a naplní výstupní parametr `signal`, v případě, že neexistuje, jeho instanci vytvoří a uloží
- `Clone` – implementace navíc ještě definuje metodu pro klonování segmentu včetně signálů

Další třídou a implementací je roura. Její diagram je možné vidět na obrázku A.4.



Obrázek A.4: UML diagram tříd pro rozhraní a implementaci roury

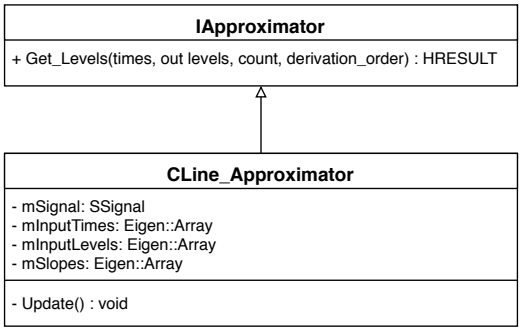
- `send` – odešle zprávu dodanou v parametru `event`, zpráva je zkopírována (volající tedy může svou instanci uvolnit z paměti); toto volání může blokovat, pokud je vnitřní fronta plná
- `receive` – vyjme zprávu z fronty čekajících a vloží ji do parametru `event`; toto volání může blokovat, pokud je vnitřní fronta prázdná
- `abort` – odblokuje všechna vlákna volající `send` nebo `receive` a odešle chybový stav. Každé další volání těchto metod nebude blokovat a bude ihned vracet chybový stav

Moduly

V této podkapitole je popsáno rozhraní jednotlivých modulů (filtrů, metrik a dalších). Každé rozhraní je ve vlastním souboru v adresáři `iface` projektu `common` spolu se vším, co je specifické pro implementace těchto rozhraní a je pro ně společné. Každé rozhraní modulu dědí od `refcnt::IReferenced` a každá implementace dědí od `refcnt::CReferenced`. Tato vazba nebude v následujících diagramech pro přehlednost dále explicitně uváděna, pokud nebude mít další, větší význam.

Aproximace a interpolace

Na diagramu A.5 lze vidět schéma rozhraní a třídy pro modul aproximace a interpolace.



Obrázek A.5: UML diagram tříd pro rozhraní a implementaci aproximace a interpolace

V současné době je podporována pouze interpolace lomenou čarou třídou `CLine_Approximator`.

- `Get_Levels` – aproximuje signál v časech `times` a výsledné hodnoty ukládá do pole `levels`, volitelně může ukládat buď signál nebo jeho derivaci, dle parametru `derivation_order`
- `Update` – modul může definovat metodu pro obnovu předpočítaných parametrů

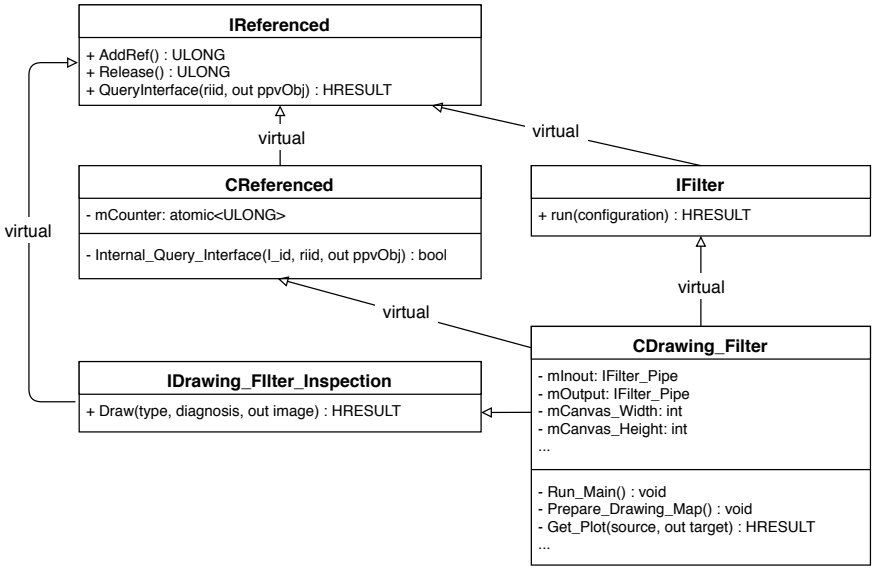
Interpolace lomenou čarou ukládá koeficienty úseček pro každý čas. Také uchovává naměřené časy a hodnoty, aby byl výpočet rychlejší.

Filtry

Rozhraní filtru `IFilter` je prosté – lze ho vidět na obrázku A.7 spolu s příkladem jednoho konkrétního filtru a hierarchie dědičnosti ostatních rozhraní a tříd.

Filtrů je velké množství a struktura všech nebude v této práci dokumentována. Čtenář je proto odkázán do zdrojového kódu, který obsahuje veškerou potřebnou dokumentaci formou komentářů výkonného zdrojového kódu a specifických funkcí a metod.

- `run` – metoda, která je vždy spouštěna řídicím kódem ve vlastním vlákně



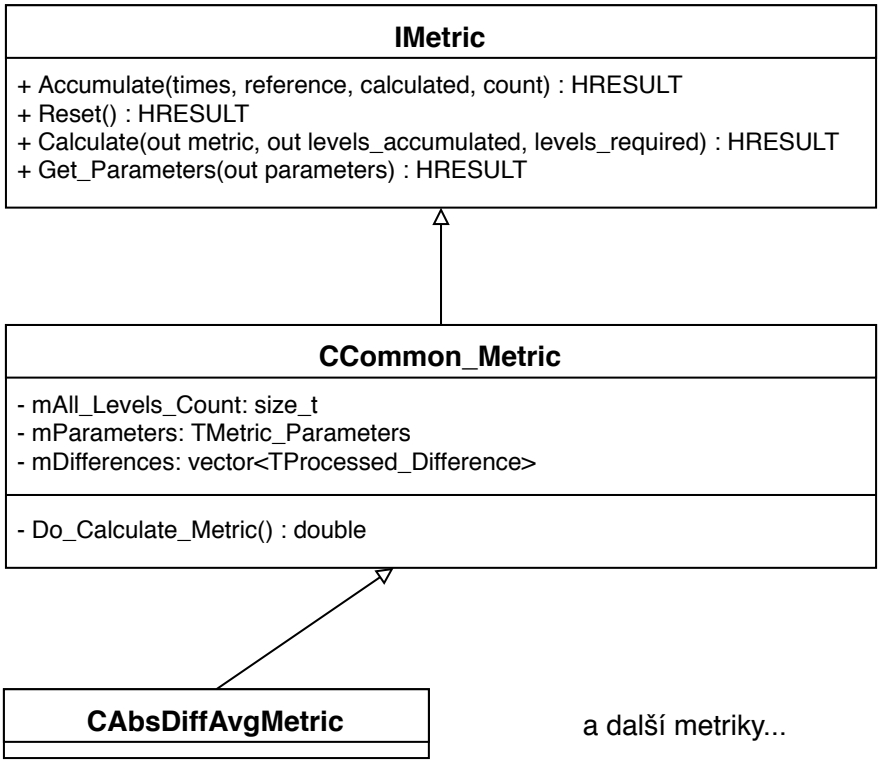
Obrázek A.6: UML diagram tříd pro rozhraní filtrů a příklad jedné implementace (jejíž atributy a metody byly výrazně zredukovány)

Metoda `run` je vždy spuštěna řídicím kódem ve vlákně, které je považováno za hlavní pro daný filtr. Filtr může spustit další vlákna, vždy je však musí ukončit před skončením vlákna hlavního. Ukončení hlavního vlákna filtru je považováno za korektní ukončení práce filtru a ten je uvolněn z paměti.

Na obrázku A.7 lze dále vidět příklad filtru pro vykreslování. Lze si všimnout, že filtr nemá žádné dodatečné veřejné metody ani atributy. To proto, že jednoduše není nutné, aby nějaké exportoval. Pro export dalších metod implementuje rozhraní `IDrawing_Filter_Inspection`. Více o dodatečných rozhraních je uvedeno ve zvláštní podkapitole.

Metriky

Na obrázku A.7 lze vidět diagram tříd a jejich dědičnosti pro metriky. Všechny metriky mají společného veřejného předka `IMetric`, který definuje základní rozhraní pro práci s metrikami. Všechny implementované metriky pak využívají společného předka `CCommon_Metric`, který `IMetric` implementuje, a navíc definuje to, co je všem metrikám společné – kontejnery pro rozdílové hodnoty, ze kterých metriky vychází. Všechny metriky jsou navrženy a implementovány tak, aby menší hodnota znamenala lepší výsledek (např. pro výpočet parametrů modelu).



Obrázek A.7: UML diagram tříd pro rozhraní metrik, společného předka implementací a příklad jedné implementace

- `Accumulate` – uloží `count` hodnot z pole časů `times`, pole referenčních hodnot `reference` a vypočtených hodnot `calculated` do vnitřních kontejnerů
- `Reset` – vyprázdní vnitřní kontejnery
- `Calculate` – vypočte hodnotu metriky a uloží ji do `metric` a počet použitých hodnot do `levels_accumulated`. Také je možné vyžádat minimální počet hodnot `levels_required`
- `Get_Parameters` – naplní parametr `parameters` parametry metriky
- `Do_Calculate_Metric` – ve společném předkovi se nachází ještě metoda pro výpočet metriky

Modely

Každý model je pouze speciální implementací signálu, tedy rozhraní `ISignal`, a pro modely ještě společného předka `CCommon_Calculation`. Oproti

signálům naměřeným mají však několik odlišností:

- vždy mají referenční signál, zpravidla naměřený (není to však podmínkou)
- metody `Get_Discrete_Bounds` a `Get_Discrete_Levels` nevrací vlastnosti a hodnoty tohoto signálu, ale signálu referenčního
- parametr `parameters` při volání `Get_Continous_Levels` musí být buď platným ukazatelem na kontejner parametrů daného modelu, nebo `nullptr` – pak se nahradí parametry výchozími

Algoritmy pro hledání parametrů modelu

Narozdíl od zbylých entit, tyto algoritmy nemají jednotného předka. Jejich instance se totiž nikdy nepřenáší přes rozhraní dynamické knihovny, pracují synchronně vůči vnějšímu kódu a výsledky předávají volajícímu v definovaných strukturách.

```
struct TSolver_Setup {
        const GUID solver_id;
        const GUID signal_id;

        ITime_Segment **segments;
        const size_t segment_count;

        IMetric *metric;
        const size_t levels_required;
        const char use_measured_levels;

        IModel_Parameter_Vector *lower_bound, *upper_bound;

        IModel_Parameter_Vector **solution_hints;
        const size_t hint_count;

        IModel_Parameter_Vector **solved_parameters;

        TSolver_Progress *progress;
};
```

Výpis kódu A.1: Struktura pro nastavení algoritmu hledání parametrů

- `solver_id` a `signal_id` – identifikace algoritmu a modelového signálu

- `segments` a `segment_count` – kopie segmentů a počet segmentů; obsah se nesmí v průběhu výpočtu měnit, segment je vhodné klonovat jeho metodou `Clone`
- `metric`, `levels_required` a `use_measured_levels` – parametry metriky, požadovaný počet hodnot a požadavek na použití měřených hodnot namísto vypočtených kontinuálních
- `lower_bound` a `upper_bound` – spodní a horní mez parametrů modelu
- `solution_hints` a `hint_count` – parametry z předchozích výpočtů, které budou použity pro odhad počátečního řešení
- `solved_parameters` – kontejner, do kterého je uloženo výsledné řešení (vypočtené parametry)
- `progress` - struktura, která je v průběhu výpočtu aktualizována o průběh algoritmu

Dodatečná rozhraní modulů

Pro potřeby simulace je nutné, aby každý modul mohl poskytnout dodatečné rozhraní, kterým je možné vnějším kódem modul dotazovat na další funkcionalitu. Typickým příkladem je již zmíněný modul – filtr pro vykreslení grafů.

Pro tuto funkcionalitu je využita metoda rozhraní `IReferenced` s názvem `QueryInterface`. Do té vstupuje identifikátor GUID daného rozhraní a parametr, do kterého bude uložen ukazatel na požadované rozhraní v případě úspěchu. Identifikátor GUID daného rozhraní je odlišný od identifikátoru GUID samotné entity – filtr `CDrawing_Filter` a dodatečné rozhraní `IDrawing_Filter_Inspection` mají tedy každé přiřazený jiný identifikátor GUID.

V současné době jsou podporována 4 dodatečná rozhraní:

- `IError_Filter_Inspection` – pro export chybových metrik do uživ. rozhraní
- `IDrawing_Filter_Inspection` – pro export vykreslených grafů do uživ. rozhraní

- `IDevice_Filter_Export` – pro možnost do architektury vkládat měření z uživ. rozhraní (např. mobilní telefon a reálný senzor)
- `IInterop_Export_Filter_Inspection` – pro možnost z architektury exportovat zprávy do uživ. rozhraní (např. mobilní telefon)

Rozhraní dynamických knihoven

Každá dynamická knihovna obsahující jakýkoliv modul musí příslušným souborem definic (soubor s příponou `def`, který linkovací nástroj používá) exportovat specifickou sadu funkcí. Tyto funkce musí být vždy exportovatelné a musí být vynechány z *name mangling* procesu[1] – v případě jazyka C++ musí být tedy uvozeny konstrukcí `extern "C"`. Dále musí dodržovat volací konvenci na systémech, kde je to vyžadováno – postačí využít makro `IfaceCalling`. Posledním obecným požadavkem je návratová hodnota typu `HRESULT`.

Každý modul musí mít tzv. deskriptor. Tyto deskriptory jsou jednotnou strukturou pro popis daného modulu, jeho názvu, parametrů a vlastností.

Aproximace a interpolace

Knihovna exportující modul aproximace a interpolace musí exportovat funkce:

- `do_get_approximator_descriptors(glucose::TApprox_Descriptor **begin, glucose::TApprox_Descriptor **end)` – funkce pro získání dostupných deskriptorů aproximačních a interpolačních modulů, vždy jde o kontinuální paměť od `begin` do `end` (bez)
- `do_create_approximator(const GUID *approx_id, glucose::ISignal *signal, glucose::IApproximator **approx, glucose::IApprox_Parameters_Vector* configuration)` – funkce pro vytvoření instance modulu aproximace nebo interpolace s daným identifikátorem GUID `approx_id`, signálu `signal` a s parametry `configuration`; instance je v případě úspěchu uložena do parametru `approx`

[1] proces, který používá například překladač jazyka C++ pro zajištění unikátnosti symbolu

```
struct TApprox_Descriptor {
        const GUID id;
        const wchar_t *description;
        const size_t parameters_count;
        const wchar_t** ui_parameter_name;
        const wchar_t** config_parameter_name;
};
```

Výpis kódu A.2: Struktura popisovače (deskriptoru) modulu aproximace a interpolace

Ve výpisu kódu A.2 je vidět popisovač modulu aproximace a interpolace.

- `id` – identifikátor GUID modulu
- `description` – název (krátký popis) modulu
- `parameters_count` – počet parametrů, které modul vyžaduje v kontejneru při vytváření
- `ui_parameter_name` – název parametru zobrazený v uživatelském rozhraní
- `config_parameter_name` – textový identifikátor parametru, kterým bude označen v konfiguračním souboru

Všechny parametry tohoto modulu jsou typu `double`, není tedy explicitně vyžadováno označení datových typů.

Filtry

Knihovna exportující modul filtru musí exportovat funkce:

- `do_get_filter_descriptors(glucose::TFilter_Descriptor **begin, glucose::TFilter_Descriptor **end)` – funkce pro získání dostupných deskriptorů filtrů, vždy jde o kontinuální paměť od `begin` do `end` (bez)
- `do_create_filter(const GUID *id, glucose::IFilter_Pipe *input, glucose::IFilter_Pipe *output, glucose::IFilter **filter)` – funkce pro vytvoření instance filtru s daným identifikátorem

GUID `id`, vstupní a výstupní rourou `input` a `output`; filtr je v případě úspěchu uložen do parametru `filter`

```
struct TFilter_Descriptor {
        const GUID id;
        const wchar_t *description;
        const size_t parameters_count;
        const NParameter_Type* parameter_type;
        const wchar_t** ui_parameter_name;
        const wchar_t** config_parameter_name;
        const wchar_t** ui_parameter_tooltip;
};
```

Výpis kódu A.3: Struktura popisovače (deskriptoru) filtru

Ve výpisu kódu A.3 je vidět popisovač filtru.

- `id` – identifikátor GUID filtru
- `description` – název (krátký popis) filtru
- `parameters_count` – počet parametrů, které filtr vyžaduje v kontejneru při volání `run`
- `parameter_type` – datový typ parametru (hodnota výčtového typu)
- `ui_parameter_name` – název parametru zobrazený v uživatelském rozhraní
- `config_parameter_name` – textový identifikátor parametru, kterým bude označen v konfiguračním souboru
- `ui_parameter_tooltip` – dodatečná informace, která se zobrazuje v uživatelském rozhraní v informační bublině

Pole `parameter_type`, `ui_parameter_name`, `config_parameter_name` a `ui_parameter_tooltip` musí mít vždy délku `parameters_count` a na odpovídajících indexech mít informace pro jeden konkrétní parametr.

Metriky

Knihovna exportující modul metriky musí exportovat funkce:

- `do_get_metric_descriptors(glucose::TMetric_Descriptor **begin, glucose::TMetric_Descriptor **end)` – funkce pro získání dostupných deskriptorů metrik, vždy jde o kontinuální paměť od `begin` do `end` (bez)
- `do_create_metric(const glucose::TMetric_Parameters *parameters, glucose::IMetric **metric)` – funkce pro vytvoření instance metriky s danými parametry `parameters`; instance je v případě úspěchu uložena do parametru `metric`

```
struct TMetric_Descriptor {
        const GUID id;
        const wchar_t *description;
};
```

Výpis kódu A.4: Struktura popisovače (deskriptoru) metriky

Ve výpisu kódu A.4 je vidět popisovač metriky.

- `id` – identifikátor GUID metriky
- `description` – název (krátký popis) metriky

Metrika je navíc konfigurována jednotnou strukturou uvedenou ve výpisu kódu A.5

```
struct TMetric_Parameters {
        const GUID metric_id;
        const unsigned char use_relative_error;
        const unsigned char use_squared_differences;
        const unsigned char prefer_more_levels;
        const double threshold;
};
```

Výpis kódu A.5: Struktura kontejneru parametrů metriky

- `metric_id` – identifikátor GUID metriky

- `use_relative_error` – má být použita relativní odchylka?
- `use_squared_differences` – má být použita druhá mocnina odchylky?
- `prefer_more_levels` – má být odchylka vydělena počtem hodnot?
- `threshold` – parametr konkrétních metrik, volající je vždy zodpovědný za nastavení hodnoty, která odpovídá dané metrice

Modely

Knihovna exportující modul modelu dynamiky glukózy musí exportovat funkce:

- `do_get_model_descriptors(glucose::TModel_Descriptor **begin, glucose::TModel_Descriptor **end)` – funkce pro získání dostupných deskriptorů modelů, vždy jde o kontinuální paměť od `begin` do `end` (bez)
- `do_create_calculated_signal(const GUID *calc_id, glucose::ITime_Segment *segment, glucose::ISignal **signal)` – funkce pro vytvoření instance vypočteného signálu s daným identifikátorem GUID `calc_id`, na segmentu `segment`; instance je v případě úspěchu uložena do parametru `signal`

```
struct TModel_Descriptor {
        const GUID id;
        const wchar_t *description;
        const wchar_t *db_table_name;

        const size_t number_of_parameters;
        const NModel_Parameter_Value *parameter_types;
        const wchar_t **parameter_ui_names;
        const wchar_t **parameter_db_column_names;

        const double *lower_bound;
        const double *default_values;
        const double *upper_bound;

        const size_t number_of_calculated_signals;
        const GUID* calculated_signal_ids;

        const wchar_t **calculated_signal_names;
```

```
        const GUID* reference_signal_ids;
};
```

Výpis kódu A.6: Struktura popisovače (deskriptoru) modelu

Ve výpisu kódu A.6 je vidět popisovač modelu.

- `id` – identifikátor GUID modelu
- `description` – název (krátký popis) modelu
- `db_table_name` – název tabulky v databázi, která má být použita pro uchovávání parametrů tohoto modelu
- `number_of_parameters` – počet parametrů modelu
- `parameter_ui_names` – názvy parametru zobrazené v uživatelském rozhraní
- `parameter_types` – interpretace parametrů (všechny parametry jsou typu `double`, lze však změnit jejich interpretaci)
- `parameter_db_column_names` – názvy sloupců tabulky v databázi pro každý parametr modelu
- `lower_bound` a `upper_bound` – spodní a horní meze pro parametry
- `default_values` – výchozí hodnota parametrů modelu
- `number_of_calculated_signals` – počet signálů, které je model schopný vypočítat
- `calculated_signal_ids` – identifikátory GUID vypočtených signálů
- `calculated_signal_names` – názvy vypočtených signálů
- `reference_signal_ids` – identifikátory GUID referenčních signálů pro vypočtené signály na odpovídajících indexech

Algoritmy pro hledání parametrů modelu

Knihovna exportující modul algoritmu pro výpočet parametrů modelu musí exportovat funkce:

- `do_get_solver_descriptors(glucose::TSolver_Descriptor **begin, glucose::TSolver_Descriptor **end)` – funkce pro získání dostupných deskriptorů modulů s algoritmy pro výpočet parametrů, vždy jde o kontinuální paměť od `begin` do `end` (bez)
- `do_solve_model_parameters(const glucose::TSolver_Setup *setup)` – funkce pro výpočet parametrů modelu, parametrizovaná kontejnerem `setup`

```
struct TSolver_Descriptor {
        const GUID id;
        const wchar_t *description;

        const bool specialized;
        const size_t specialized_count;
        const GUID *specialized_models;
};
```

Výpis kódu A.7: Struktura popisovače (deskriptoru) modulu pro výpočet parametrů modelu

Ve výpisu kódu A.7 je vidět popisovač modulu pro výpočet parametrů modelu.

- `id` – identifikátor GUID modulu algoritmu
- `description` – název (krátký popis) algoritmu
- `specialized` – je tento algoritmus specializovaný? (viz níže)
- `specialized_count` – počet modelů, které je algoritmus schopný uvažovat
- `specialized_models` – identifikátory GUID modelů, které je algoritmus schopný uvažovat

Algoritmy se dělí na dvě skupiny dle hodnoty parametru `specialized`:

- *specializované* – umí počítat parametry jen některých modelů, jejichž výčet uvádí popisovač
- *nespecializované* – umí počítat parametry libovolného modelu

Ostatní

Posledním možným exportem je export funkce pro vytváření naměřeného signálu.

- `do_create_measured_signal(const GUID *id, glucose::ITime_Segment *segment, glucose::ISignal **signal)` – funkce pro vytvoření instance naměřeného signálu s daným identifikátorem GUID `id`, na segmentu `segment`; instance je v případě úspěchu uložena do parametru `signal`

Tato funkce je implementována modulem `signals` a podporuje všechny doposud známé a používané naměřené signály s identifikátory:

- `glucose::signal_IG` – signál naměřený v intersticiální tekutině
- `glucose::signal_BG` – signál naměřený v krvi
- `glucose::signal_Insulin` – signál pro podávané množství inzulinu
- `glucose::signal_Carb_Intake` – signál pro příjem karbohydrátů
- `glucose::signal_Calibration` – signál pro kalibraci senzoru signálem z krve

Uživatelské rozhraní

Řešení pak obsahuje i další tři části, ve kterých je implementováno uživatelské rozhraní pro požadované platformy:

- `console` – konzolová aplikace pro libovolnou platformu, ale bez ovládacích prvků
- `desktop` – aplikace pro osobní počítač
- `mobile` – mobilní aplikace pro mobilní telefon s operačními systémy Android a iOS

Společný kód

Pro konzolovou aplikaci a aplikaci pro osobní počítač jsou společným kódem zejména čtyři části implementace:

- dynamická knihovna `factory` – stará se o vyhledání dynamických knihoven, vyhledání předdefinovaných symbolů z jejich rozhraní a uložení příslušných funkcí pro další použití; jejím voláním jsou získávány dostupné deskriptory a vytvářeny modulové instance
- `CConfig` – třída pro správu konfigurace, načítá konfigurační soubor `config.ini`, z něj načítá příslušné filtry a jejich identifikátor a konfiguraci ukládá do instance třídy `CFilter_Chain`
- `CFilter_Chain` – třída obsahující „řetěz“ filtrů v pořadí, ve kterém byly definovány v konfiguračním souboru; každý filtr má zde uloženou i konfiguraci
- `CFilter_Chain_Holder` – třída pro instancování filtrů, udržování jejich instancí a korektní ukončování filtrů při dokončení jejich práce

Konzolová aplikace

Konzolová aplikace obsahuje pouze malý zdrojový soubor `main.cpp`, který spouští společný kód tříd `CConfig` a `CFilter_Chain_Holder`.

Tento kód pouze načte konfigurační soubor `config.ini`, spustí simulaci a čeká na její skončení – až uživatel stiskne kombinaci kláves `ctrl-C` (signál `SIGINT`).

Aplikace pro osobní počítač

Aplikace pro osobní počítač využívá knihovny *Qt* pro tvorbu uživatelských rozhraní a další funkce, jako je například přístup do databáze.

Rozhraní používá princip MDI[2]. Aplikace obsahuje jedno hlavní okno, v

[2]Multiple documents interface, rozhraní pro editaci většího množství dokumentů nebo obsažení více pohledů

rámci něhož může být otevřeno několik oken dalších. Lze je na sebe skládat, překrývat a přeuspořádat dle vzoru (horizontálně, vertikálně).

Přehled tříd oken:

- `CMain_Window` – hlavní okno, *MDI parent*, definuje klasické výsuvné menu
- `CFilters_Window` – okno pro konfiguraci filtrové architektury, zde je možne přidávat, ubírat, přesouvat a konfigurovat filtry; nejvýše jedna instance za běhu aplikace
- `CFilter_Config_Window` – okno pro konfiguraci konkrétní instance filtru, zde se nastavují veškeré parametry v předem připravených editačních widgetech
- `CSimulation_Window` – okno simulačního pohledu, zde se spouští, krokuje a ovládá celá simulace, jsou zde vidět výstupy v podobě grafů, chybových metrik a záznamu; nejvýše jedna instance za běhu aplikace

Podpůrné kontejnery uživatelského rozhraní:

- `CSelect_Time_Segment_Panel` – záložka konfigurace filtru pro výběr časových segmentů z databáze
- `CModel_Bounds_Panel` – záložka pro zadávání spodní a horní meze parametrů vybraného modelu a výchozích parametrů

Záložky v simulačním pohledu:

- `CDrawing_Tab_Widget` – záložka, která zobrazuje na své ploše vykreslený graf; je možné parametrizovat zejména typem grafu
- `CErrors_Tab_Widget` – záložka s chybovými metrikami uspořádanými v tabulce
- `CLog_Tab_Widget` – záložka se záznamem ze simulace

Konfigurace filtrů je učiněna v rámci instance okna `CFilter_Config_Window`, jehož obsah je generován dynamicky podle deskriptoru filtru a jeho parametrů, datových typů a popisků.

Každý datový typ má definován svůj ovládací prvek pro vstup hodnoty. Všechny tyto prvky dědí od jednotného předka `CContainer_Edit`. Ten definuje metody pro přístup:

- `get_parameter` – pro získání zadané hodnoty; vždy je validováno
- `set_parameter` – pro zapsání hodnoty, např. při inicializaci widgetu
- `apply` – po stisku tlačítka pro aplikaci změn; implementace je nepovinná

Dle typu vstupu existují definované kontejnery:

- `CWChar_Container_Edit`, `CDouble_Container_Edit`, `CInteger_Container_Edit` a `CBoolean_Container_Edit` – vstupy pro primitivní datové typy nebo řetězce
- `CGUID_Entity_Combobox` – výběrový box pro všechny typy entit, které lze označit identifikátorem GUID, a existuje pro ně jednoduchá funkce, která vrací vektor se seznamem desktiptorů
- `CModel_Signal_Combobox` – výběrový box pro výběr signálu; vždy je propojen se vstupem pro výběr modelu
- `CAvailable_Signal_Select_Combobox` – výběrový box pro výběr libovolného signálu, který je dostupný

Aplikace dále definuje dva speciální filtry – GUI filtr a vstupní filtr. GUI filtr zaobaluje funkcionalitu `CDrawing_Filter`, `CErrors_Filter` a `CLog_Filter` a sdružuje jejich používání do jednoho filtru dostupného výhradně z aplikace pro osobní počítač. Tím, že jde o speciální filtr nemusí být dodržovány konvence pro rozhraní dynamických knihoven a obě entity mohou spolu komunikovat přímo, což dovoluje efektivnější komunikaci při výměně dat, jejichž generování je podmíněno nějakou zprávou procházející filtrovou architekturou. GUI filtr se pak stará i o korektní ukončení všech zaobalených filtrů.

Vstupní filtr slouží jako filtr na začátku celé architektury pro možnost odesílání zpráv do všech filtrů. Typicky je takto řízen chod simulace, její krokování, pozastavení nebo spuštění algoritmů pro výpočet parametrů modelu, a další.

Mobilní aplikace

Použití technologie Xamarin, Xamarin.Forms a sdíleného kódu vyžaduje oddělit implementace společné a specifické části do více projektů. Je proto definováno dělení na:

- `gpredict3_mobile` – sdílený kód; zde je většina funkcionality a uživ. rozhraní
- `gpredict3_mobile.Android` – kód specifický pro platformu Android
- `gpredict3_mobile.iOS` – kód specifický pro platformu iOS

K mobilní aplikaci je také připraveno sestavení knihovny Intel TBB a NLopt. Oba tyto projekty se ke všem knihovnám, které je vyžadují, linkují staticky. To z důvodu omezení, která mobilní platformy na aplikace kladou.

Všechny knihovny jsou kompilovány pro OS Android jako dynamické a pro iOS jako statické, opět kvůli omezení dané platformy. V případě OS Android činí výjimku implementace obdoby knihovny `factory`, která je linkována ke všem ostatním knihovnám také staticky.

Aplikace definuje tyto pohledy:

- `MainPage` – hlavní stránka, která je dělena na záložky s hodnotami, grafy a nastavením senzoru
- `SensorSelectPage` – stránka pro výběr senzoru z dostupných nalezených
- `SensorDetailsSelectPage` – stránka pro potvrzení výběru senzoru
- `BloodInput` – část pohledu pro dialog zadávání hodnoty koncentrace glukózy z krve

- `BloodInputSelect` – část pohledu pro výběr typu zadané koncentrace (kalibrace nebo dodatečné měření)

Podpora senzorů je zaručena generickou třídou `DeviceAdapterBase` a její specializací pro konkrétní senzor. V současné době je definován pouze jeden typ podporovaného zařízení ve třídě `IEEEDevice` – implementace senzoru dle standardu IEEE 11073.

Hodnoty jsou pak exportovány pomocí registrace „posluchače“ implementujícího `DeviceValueListener`.

Seznam zkratek

AIC Akaike information criterion – metrika používaná pro ohodnocení kvality modelu

ATT Attribute protocol – protokol pro správu atributů standardu BLE

BG Blood glucose – koncentrace glukózy v krvi

BLE Bluetooth-Low Energy – standard Bluetooth pro nízkopříkonová zařízení

CGMS Continuous Glucose Measurement System – systém pro nepřetržité měření hladiny glukózy v podkoží

FHSS Frequency hopping spread spectrum – metoda přenosu v rozprostřeném spektru

GAP Generic access profile – protokol jednotného přístupu standardu BLE

GATT Generic attribute profile – profil jednotných atributů standardu BLE

HCI Host-controller interface – rozhraní hostitelské a řídicí části architektury standardu BLE

IDF International diabetes federation – Mezinárodní diabetická federace

IEEE Institute of Electrical and Electronics Engineers – organizace zaštit'ující vybrané technické standardy

ISIG Interstitiary signal – signál získaný měřením v podkožní tkáni

IG Interstitiary glucose – koncentrace glukózy v podkožní tkáni

PAN Personal area network – sít' v osobním prostoru člověka

RACP Record access control point – charakteristika BLE pro získávání specifických hodnot měření

WHO World Health Organization – Světová zdravotnická organizace

Seznam obrázků

Obsah přiloženého CD

- `bin` - zkompilovaný program
  - `desktop` - aplikace pro osobní počítač (MS Windows)
  - `mobile` - aplikace pro mobilní telefon (Android APK)
- `sources` - zdrojové soubory programu
  - `desktop` - aplikace pro osobní počítač
  - `mobile` - aplikace pro mobilní telefon
  - `sensor` - firmware simulovaného senzoru (IEEE 11073)
- `thesis` - text této práce včetně zdrojových souborů nástroje LaTeX

Ochranné známky

Bluetooth® je registrovaná ochranná známka organizace Bluetooth SIG, Inc.

Wi-Fi® je registrovaná ochranná známka organizace Wi-Fi Alliance.

Microsoft®, Windows® a příslušná loga jsou registrované ochranné známky společnosti Microsoft Corporation v USA a dalších zemích.