**Usnadnění vývoje malých senzorových zařízení**

Cíl: Připravit pomocné / vzorové programové části pro snadnou a rychlou tvorbu malých senzorových zařízení zejména pro experimentální / výzkumné / medicínské účely obsahující běžně dostupné senzory a komunikující pomocí běžně dostupných kanálů pro přenos naměřených dat.

Části práce (rozepsáno podle postupu práce, ne podle bodů v oficiálním zadání):

* Vybrat několik příkladových a často využívaných senzorů / aktuátorů připojovaných pomocí komunikační kanálů zejména na krátkou vzdálenost: pohybový senzor, teploměr, vlhkoměr, spínače, potenciometry, …
* Vytvořit vzorové / základní části programového kódu (formou knihovny) pro senzorový modul pro vybrané senzory a jejich komunikační kanály / sběrnice, například: digitál vstup, analog vstup, I2C, SPI, 1Wire, SerialPort, … (Rozdělit programový kód na úroveň komunikační a úroveň senzorovou.)
* Jako senzorový modul použít dvě nejrozšířenější zařízení a to například ARM typu STM32 (podle výběru / dostupnosti, ne formou Arduino, ale pomocí HAL knihoven a programování s ST-Link/J-Link) a nějaký typ Arduino (ideálně ESP8266/32). Tvorba pomocí C (STM32 + HAL) nebo C++ (ESP8266/32).
* Navrhnou několik základních formátů pro ukládání dat ze senzorů pro jejich snadný přenos do centrálního zařízení: bool, Byte, Int16, string, byte[], …
* Navrhnout a vytvořit protokoly pro běžné komunikační kanály využívané i na větší vzdálenosti TCP/IP(LAN/WIFI), UDP(LAN/WIFI), BlueTooth Classic/LE, USB-HID, RS232 za účelem přenosu dat ze senzorového modulu do (nějaké) centrální jednotky (počítač, tablet, telefon) Za tímto účelem vytvořit programový kód (knihovnu) umožňující přenos dat (pouze) ze dříve navržených formátů pomocí těchto různých komunikačních kanálů a to pokud možno pouhou změnou nastavení / konfigurací.
* Na centrálním zařízení (počítač, tablet, telefon) navrhnout a vytvořit způsob ukládání přijatých dat (všech dříve navržených formátů). Využít .NET Core C#.
* Vytvořit základní aplikaci na centrálním zařízení pro kontrolu přijatých / uložených dat (stačí pouze textové zobrazení formou tabulek). Využít .NET Core C#.

Poznámky:

* Vybrat pouze některé senzory (zhruba 5). Využít i senzory se stejnými komunikačními kanály (jeden kód, více příkladů použití).
* Vybrat pouze některé lokální sběrnice (max 5). Využít i ty nejjednodušší jako je digitální / analogový vstup (minimum programové práce, přesto velmi potřebné). Všechny jsou velmi často na WWW dosti popsané.
* Navrhnout jen základní formáty dat ze senzorů, na které lze převést naprostou většinu měřených (základních) hodnot, například pouze: bool, Uint8, Int16, Int32, Int16p2, Int32p2, Float, string, byte[].
* Vybrat v podstatě jakékoli STM32-NUCLEO64 (ideálně nějaké silnější STM32L4/STM32F4, příklady kódu kdekoli po WWW). Arduino ESP8266/32 obsahuje mnoho RAM a FLASH, tedy snadný vývoj programu. Obsahuje přímo WIFI/BlueTooth (velká programová podpora na WWW).
* Pro komunikaci na větší vzdálenost vybrat kanály podle vhodnosti použitého procesoru:
  + STM32 – RS232, HID-USB, BlueTooth Classic/LE, (LAN?)
  + ESP8266 – RS232, WIFI, BlueTooth Classic/LE, (LAN?)
* Přenosový protokol navrhnout co nejjednodušší
  + zcela textový (řetězec, XML, JSON) – RS232, BlueTooth Classic, (LAN?)
  + binární (nějaká komprese dat) – HID-USB, BlueTooth LE
* Pro uložení dat v centrálním zařízení převést všechny formáty dat ze senzorového modulu na minimum základních typů v .NETu, například: bool, Int32, double, string, byte[] (mohlo by stačit). Uchovávat i datum/čas posledního přijmu dat pro detekci selhání přenosu dat.
* Pro demonstrační aplikaci využít .NET Core C#. Lze snadno vytvořit (skoro stejnou) aplikaci pro PC (MS/Windows) i tablet/mobil (Google/Android). Apple/iOS pouze v případě dobré znalosti (nutná licence pro tvorbu aplikací). Základní projekt pro MS/Windows a Google/Android je jiný, ale vše co není GUI (a některé komunikace) je zcela stejné, tedy zdrojové soubory linkované do obou projektů. Základní projekty a všechny potřebné komunikace pro MS/Windows a Google/Android dodá vedoucí práce (pokud bude potřeba). GUI (WPF, Xamarin.Forms) formou základního zobrazení a tabulek je v MS/Windows a Google/Android stejné tak na 80%.

Výsledek je určen pro:

* Mírně/středně pokročilé uživatele se základní znalostí programování procesorů a počítačů.
* Tvorbu jednoduchých modulů se senzory a přenosem dat do PC/tabletu/telefonu.
* Přenos relativně malého / omezeného množství dat (max. KB/s).
* Zobrazení / zpracování dat na běžně dostupných zařízeních PC/tabletu/telefonu.
* Určeno převážně pro rychlé sestavené HW/SW pro experimenty (měřící metody, pokusná zařízení, experimentální hračky, diagnostické/medicínské senzory, …).
* Jde zejména o jednoduché projekty / zařízení, ne žádné sofistikované.

Dnes je velmi složité vybrat vhodný (nikoli zcela správný, to asi nelze) HW a SW. Z důvodu nepřeberného množství zejména jednodeskových HW (skupiny nadšenců a firmy snažící se protlačit svá řešení) je v tomto velmi složitá orientace. Pro výběr vhodného HW / SW platí tato pravidla (uvažováno nasazení zejména pro experimenty, nikoli mnoha-kusovou výrobu):

* Použít co nejdostupnější a nejrozšířenější typ HW, operačního systému, programovacího jazyka.
* Použít HW raději od firmy co vyrábí mnoho typů a varant HW.
* Použít zejména to co používají lidé kolem.
* Použít to co lze použít na velmi vysoké a současně i na velmi nízké úrovni.
* Použít pouze to na co jsou velmi dobře dostupné programovací / ladící nástroje (nejlépe aspoň ty základní zdarma).
* Použít zejména to za čím stojí velké firmy / komunity (vyvarovat se projektů nadšenců).
* Používat minimum různých technologií, aby se v tom další lidé vyznali.
* Vytváříme projekt pro uživatele ne pro sebe.

Příklady:

* Pokud udělám program v C/C++ tak je přenesu na skoro všechny základní procesory i když tento program zabere trochu více práce. Když použiji Python/LUA/… jsem omezen interpretem tohoto jazyka na cílovém procesoru. Pro malý pokus použiji Python/LUA/…, potřebuji více (výkon, silnější procesor), tam není Python/LUA/…, vše zahodit a udělat znova. Tomto je potřeba se vyhnout.
* Pro časté systémy MS/Windows a Google/Android použít něco velmi dobře přenositelného jako je .NET Core. Mohu použít pro WIN třeba C/C++, pro Android použít JAVA/Kotlin a pro iOS třeba Objective-C, pak docílím zmatku pro nástupce při úpravě cílové aplikace na různá zařízení. Dopadne to stejně jako na Linuxu, každý se to raději udělá znova (pro sebe). Tomuto je potřeba se vyhnout.
* Vzorový Linuxový přístup je Arduino (trochu upravený jazyk, žádná podpora laděni, terminál nepovažuji za ladění). Pro jednoduchou věc použiji Arduino, nějak to splácám dohromady a rychle to chodí. Jakmile potřebuji něco složitějšího tak třeba Arduino pro tento typ ARMu není a musím (skoro) vše předělat. Arduino nemá HW podporu ladění, pokud se zařízení zesložití a najednou jej potřebuji ladit, musím opustit Arduino a začít od (skoro) začátku. Tomuto je potřeba se vyhnout. (Zajímavost: Překlad pro Arduino má vždy zapnuté DEBUG informace, čímž se značně zvětšuje objem kódu, ale přitom Arduino je vůbec nepoužívá, kdo toto dělal?)

Proč se vybral právě tento HW / SW:

* STM23 jsou velmi rozšířené ARM procesory. Jsou dostupné v mnoha běžných verzích CortexM0/M3/M4/M7 a řadách STM32L/STM32F/STM32G
  + Pomocí knihoven HAL je SW velmi snadno přenositelný na různé typy STM32 (minimum úprav).
  + Přibývají verze SoC (SystémOnChip) s BlueTooth a WIFI.
  + Zdarma vývojové nástroje STM32CubeIDE. Založeno bohužel na Eclipse-IDE, ale celkem použitelné. (Oproti MS Visual Studio Community, které je také zdarma je toto návrat do doby kamenné, ale lze to snést).
  + Velmi dostupné a velmi levné vývojové desky NUCLEO/DISCOVERY/…
  + Vývojové desky (skoro všechny) obsahují programátor/debugátor v hodnotě několika tisíc korun. Zcela použitelné i pro programování externích procesorů. Možnost libovolného ladění programu (zastavení, registry, …).
  + Dostupné knížky a mnoho ukázek programového kódu po WWW.
  + Využívá je mnoho firem do svých zařízení. Velká podpora diskuzí na WWW.
  + Dnes moc nemá význam používat 8bit nebo neARM procesory, zejména pro obecná a málo-kusová zařízení.

(Určitě existují lepší ARMy, například NXP nebo AnalogDevice, možná FreeScale, ale v žádném případě se nevyznačují takovouto snadnou dostupností / cenou a podporou pro vývoj.)

* Arduino zejména mezi nadšenci velmi oblíbené (pro seriózní věci zásadně nedoporučuji). Využijeme ESP8266/32:
  + Dostatek paměti FLASH/RAM pro program (na rozdíl od například Atmel-AVR).
  + Integrované WIFI (ESO8266), případně i BlueTooth (ESP32).
  + Podpora v Arduino-IDE a knihovny skoro pro cokoli (Arduino-IDE je spíše poznámkový blok nikoli IDE, vzorový linux přístup – znova a podle sebe když je již mnoho IDE, nyní se však dělá nějaká nová verze).
  + Nemožnost ladění programu, případně pomocí textového terminálu (bohužel linuxový styl přístupu, i když nějaké náznaky zlepšení již jsou).
* C (STM32) / C++ (ESP) velmi rozšířený jazyk pro procesory:
  + Využitelná v podstatě na všech typech procesorů.
  + Velká přenositelnost kódu (nikoli volaných funkcí z knihoven výrobce).
  + Nepoužívat (často pomalé) skriptovací jazyky Python/LUA/ … Občas je nutno nastavovat parametry (sběrnice/registry) v HW a ty nemusí být přes skriptovací jazyky dostupné. Rovněž tyto jazyky nejsou pro širokou část procesorů.
* PC (MS/Windows) a tablet/mobil (Google/Android):
  + Nejčastější požívaná zařízení PC/tablet/mobil obsahují tyto systémy. Velmi vhodné pro experimenty/pacienty/doktory/….
  + Velmi snadný vývoj aplikací pomocí vysoko-úrovňových jazyků a přenositelnost aplikace (i s GUI). V podstatě jedny WIN, jeden Adnroid (pouze vývojové verze). Ne jako u Linuxu mnoho variant a vznikají stále nové (mizerná kompatibilita).
  + V podstatě jistá dostupnost zařízení s MS/Windows a Google/Android na cílovém pracovišti. Apple/iOS nevhodný pro experimenty (nemožnost snadné instalace na libovolné zařízení za účelem rychlého experimentu pro pacienty/doktory).
* .NET Core dnes již celkem dobrá (OpenSource) multiplatformní technologie pro vývoj aplikací:
  + Vývoj aplikací (nejen) pro MS/Windows (standardní, IoT, WinRT, …), Google/Android a Apple/iOS.
  + Velká přenositelnost kódu (ne GUI kód skoro 100%, GUI podle použití 50-80%, po vydání nového MAUI, podzim 2021 bude ještě větší) případě forma knihoven pro daný typ systému.
  + Vysoko-úrovňový rychlý a bohatý framework (ne jako JAVA) s možností i velmi nízkého programování a přístupu do operačního systému (volání jeho funkcí).

Spousta informací vhodných pro tvorbu zadání bude postupně na adrese:

<http://cyber.felk.cvut.cz/novakpe/Information/>

Možnosti / doporučení:

* Lze použít i třeba Raspberry PI (3 a víše) s MS/Windws IoT a .NET C# pokud by byl požadavek, pro GUI (přes HDMI) je vhodná Avalonia, skoro věrná kopie WPF.
* Jako centrální zařízení lze použít v podstatě cokoli, na čem běží systém programově / aplikačně nezávislý na HW ideálně .NET Core pro MS/Windows (IoT) nebo Google/Android.
* Hlavně nepoužívat žádný Linux a jakékoli jeho odnože (v budoucnu pouze cesta do pekel z důvodu nekompatibility, viz verze Raspberry PI). Tedy ne určité distribuce Linuxu vytvářené komunitou nadšenců, bez budoucí podpory při změně HW, v podstatě všechny běžné jednodeskové nadšenecké procesorové systémy. (Google/Android je Linux za kterým stojí velká firma s nutností zpětné kompatibility, tedy pravý opak linuxu na Raspberry PI.)

Výsledek práce / ukázka:

* STM32-NUCLEO64 s několika senzory (podle výběru / možností)
* ESP8266/32 s několika senzory (podle výběru / možností)
* Připojení do PC (MS/Windows) pomocí RS232, HID-USB, BlueTooth Classic/LE, WIFI, (LAN?) (podle výběru)
* Připojení do tabletu/telefonu pomocí BlueTooth Classic/LE, WIFI
* Aplikace na PC (MS/Windows) a tablet/telefon (Google/Android) zobrazující přijímaná data.
* Ukázka konfigurace / přidání senzoru do senzorového modulu (nějakého z vybraných).
* Ukázka nastavení / přepnutí komunikačního kanálu do centrální jednotky (například mezi RS232, BlueTooth Classic/LE, WIFI).
* Ukázka zobrazení / přijmu dat na displeje centrálního zařízení při změně hodnoty ze senzoru.
* V případě potřeby / zájmu vedoucí práce poskytne již existující medicínské aplikace (MS/Windows a Google/Android) pro demonstraci reálného využití dat z připojených senzorů.

Poznámky pro programování:

* V senzorovém modulu (ARM/ESP) bych nepoužíval ve svém kódu třídy a dědičnost. Dědičnost a virtuální metody bych obešel konfigurací pomocí struktur a ukazatelů na funkce (mnoho malých procesorů nepoužívá C++).
* V centrální jednotce lze použít cokoli dostupné z .NET Core, ale omezit se na ty části, které jsou dostupné na všech platformách pro .NET Core (nepoužívat tedy nějaké vychytávky, pokud to nebude nutné).

Něco k postupu práce:

* Vybrat soubor senzorů (i těch co v práci nepoužijí) a sestavit seznam jaké typy (měřených) hodnot poskytují (bool, int, float, …), jakého rozsahu jsou čísla, kolik desetinných míst běžně, … Z tohoto sestavit / vybrat nejmenší množství typů hodnot pro uložená v senzorové jednotce.
* Navrhnout / vytvořit „struct(s)“ pro uložení těchto hodnot v senzorové jednotce. Tak by bylo možno snadno data plnit ze senzorů a rovněž automaticky vysílat do centrální jednotky.
* Navrhnout / vytvořit formát programového souboru pro připojení senzoru, tedy funkce, které musí senzor poskytovat (Init, GetValue, …). Jeden typ senzoru jeden soubor. Zde vznikne problém, že nelze vložit do projektu více stejných souborů (název, data, funkce) což lze obejít makrem přidávající do názvů například pořadové číslo.
* Navrhnout / vytvořit formát programového souboru pro jeden typ lokální komunikace, tedy funkce, které musí lokální komunikace poskytovat (Init, Send, Receive, …). Tyto funkce budou volány ze souborů programové obsluhy senzorů.
* Navrhnout / vytvořit jak zapsat konfiguraci senzorů v modulu, tedy jaká data poskytují, jak jsou za sebou data ukládána v modulu (tak budou ukládána i při přenosu), odkazy na základní funkce sensoru (zejména inicializace).
* Navrhnout formát přenosu dat do centrální jednotky. Zřejmě dva typy a to textový a binární.
* Navrhnout / vytvořit formát programového souboru pro komunikaci s centrální jednotkou. Stanovit požadované funkce pro komunikaci (Init, Send, Receive, …).
* Navrhnout / vytvořit jak zapsat konfiguraci pro komunikaci v modulu (jaký typ, jaký formát, parametry, …).
* Navrhnout / vytvořit třídy (zde raději ne struktury) pro uložení přijatých hodnot v centrální jednotce, minimální možný počet, jak převádět přijatá data na tyto typy.
* Navrhnout základní princi vyzvedávání dat z těchto datových tříd na centrální jednotce aplikací uživatele (pouze princip pro budoucí uživatele) s případným využitím událostí o vložení / přijetí nové hodnoty (okamžitá reakce na přijatou hodnotu).
* Minimalistická aplikace na MS/Windows a Google/Android periodicky / událostně vyzvedává data z datových tříd a zobrazující je v GUI. Zobrazení hodnot bude postačující formou tabulek. V případě zájmu lze použít jakékoli grafy, bargrafy, ukazatele, …
* BlueTooth Classic použít stylem RFCOM, tedy virtuální / vzdušný sériový port.

Podle bodů se zdá jako hodně práce, ale není tomu tak. Jedná se zejména o návrh a použití mnoha kódu z WWW nebo dodaného vedoucím práce.

* Pro ESP8266/32 (Arduino) jsou všechny lokální / vzdálené komunikace hotové / knihovny (zde zásadně nic nevytvářet, pouze využít). Komunikace přes WIFI (TCP/IP a UDP) je zcela dostupná. BlueTooth Clasic (případně LE) zřejmě také, seriál port (přes USB) je triviální pomocí VCP (print), jiné asi nebudou potřeba. Rovněž jsou na WWW příklady použití snad všech typů senzorů. Jde zde v podstatě pouze o návrh sktruktury souborů, konfigurace a dat.
* Pro STM32 je to trochu horší. Jsou příklady všech lokálních sběrnic, bude však nutno upravit podle vlastního návrhu. Na WWW příklady mnoha senzorů, lze použít. Komunikace po seriál port (přes USB) je triviální pomocí VCP, BlueTooth Classis se použije HC-05/06 ve stylu RFCOM, pro BlueTooth LE se použije buď STM SPBTLE-RF (X-NUCLEO-IDB05A1 využívající SPI) nebo Microchip NR4871 (využívající sériový port, obsluhu poskytne vedoucí práce).
* Na PC (MS/Windows) a tablet/telefon (Google/Android) jsou komunikace pomocí WIFI (TCP/IP a UDP) programově zcela stejné. Pro komunikace BlueTooth Clasic/LE poskytne programovou část vedoucí práce. Sériový port na PC (MS/Windows) je triviální pomocí VCP. Pokud bude použít USB-HID, tak základní komunikaci poskytne vedoucí práce (přenos po blocích jako v případě BlueTooth LE). Založené a nakonfigurované projekty pro PC (MS/Windows) a tablet/telefon (Google/Android) poskytne vedoucí práce (v případě potřeby).
* Všechny (obecné) potřebné informace budou rovněž vkládány (podle potřeby řešitele) na adresu: <http://cyber.felk.cvut.cz/novakpe/Information/>
* Informace (podle dohody) přímo k tomuto projektu / práci budou vždy na adrese: <http://cyber.felk.cvut.cz/novakpe/BPDP/Aktualni/JinaMarek/>

Naprostá většina práce vychází z BP studenta Davida Juříka, která není dotažena dokonce (jak jsem chtěl) zejména z důvodu příchodu „koronáče“ a možná i celkově velkého rozsahu práce.

<http://cyber.felk.cvut.cz/novakpe/BPDP/Hotove/JurikDavid/>

Jedná se však o popis celkové myšlenky, návrh mnoha částí, některé se upraví, některé doplní. Nechci tady popisovat znova co je v této BP. Je potřeba to prostudovat / pochopit a poposunout o kus dále (samozřejmě za podpory vedoucího). Vše je samozřejmě na vzájemné dohodě, ale již je oficiální zadání BP napsáno a je potřeba se jej tedy držet. Pokud se něco udělá jinak (použije/nepoužije) je nutno toto v textu dostatečně zdůvodnit.

Toto téma je dobrá práce, kterou lze vhodně prezentovat a má velké využití (shodne se mnoha zájemcům).