České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Lukáš Salich**

Studijní program: Softwarové technologie a management
Obor: Softwarové inženýrství

Název tématu: **Implementace ovladače chromatografu**

Pokyny pro vypracování:

Nastudujte ovládání chromatografu Hitachi Chromaster a Hitachi Primaide a obě ovládání porovnejte.
Analyzujte ovladač chromatografu Hitachi Chromaster a na jeho základě navrhněte realizaci ovladače chromatografu Hitachi Primaide. Popište aplikaci Clarity, pro kterou bude ovladač vyvíjen.
Implementujte ovladač chromatografu Hitachi Primaide pro aplikaci Clarity. Funkčnost ovladače otestujte na chromatografu Hitachi Primaide a porovnejte ji s funkčností ovladače chromatografu Hitachi Chromaster.

Seznam odborné literatury:

Dodá vedoucí práce

Vedoucí: Ing. Viktor Černý

Platnost zadání: do konce zimního semestru 2014/2015

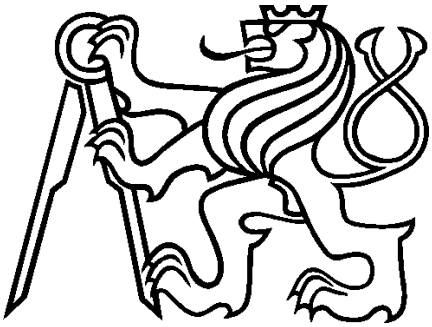
doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

L.S.

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 1. 10. 2013

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Implementace ovladače chromatografu

Lukáš Salich

Vedoucí práce: Ing. Viktor Černý

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Softwarové inženýrství

1. října 2013

Poděkování

Děkuji Viktorovi Černému za odborné vedení práce, věcné připomínky, dobré rady a vstřícnost při konzultacích a vypracovávání bakalářské práce.

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 19. 5. 2014 .....

Lukáš Salich

Anotace

Cílem práce bylo vytvoření ovladače chromatografu Hitachi Primaide pro aplikaci Clarity. Práce obsahuje popis aplikace Clarity a prostředí Clarity SDK, porovnání řízení chromatografu Hitachi Chromaster s Hitachi Primaide, analýzu ovladače pro Hitachi Chromaster, implementaci ovladače pro Hitachi Primaide a porovnání funkčnosti obou ovladačů na svých sestavách.
Ovladač pro Hitachi Primaide byl vyvinut v programovacím jazyce C++ s užitím WinAPI, MFC a Clarity SDK jako projekt v IDE Visual Studio 2010 s pomocí verzovacího nástroje Plastic SCM, výsledkem je dynamicky linkovaná knihovna, kterou lze ve Windows zaregistrovat a používat v aplikaci Clarity.

Klíčová slova

chromatografie, řídící modul

Abstract

The goal of this thesis was creating a Hitachi Primaide chromatograph driver for Clarity application. The work includes Clarity and Clarity SDK description, Hitachi Chromaster and Hitachi Primaide chromatograph control comparison, Hitachi Primaide driver implementation and functional comparison of both drivers on their respective units.
Hitachi Primaide driver was developed in C++ programming language using WinAPI, MFC and Clarity SDK as project in Visual Studio 2010 IDE with the help of Plastic SCM version control tool, the result is dynamically linked library, which can be registered in Windows and used in Clarity application.

Keywords

Clarity, chromatography, control module, DataApex, Hitachi, Visual Studio, DLL

Obsah

Úvod

Chromatografie je souhrnné označení pro skupinu fyzikálně-chemických oddělujících metod. Oddělování je založeno na rozdílné distribuci složek směsi v zařízení zvaném chromatograf. Analytická chromatografie slouží ke zjištění existence složky vzorku a k určení její koncentrace ve vzorku. Kapalinová (rozdělovací) chromatografie (LC) používá pro přenos vzorku kapalinu, tím se liší od plynové chromatografie (GC).

Aplikace Clarity je jedno z nejpoužívanějších softwarových řešení pro analytickou chromatografii, řídící přes 400 chromatografických sestav. Hitachi je velkým výrobcem chromatografických sestav a v posledním roce uvedlo na trh sestavy Chromaster a Primaide.

Obě zařízení používají kapalinovou chromatografii (LC), obě jsou složena z modulů (pumpa, automatický vzorkovač, termostat a několik detektorů), ale liší se ve vrstvách pomocí kterých komunikují přes USB, liší se metodami použitými ke komunikaci s těmito vrstvami a liší se v některých situacích způsobem, jakým reagují na změnu vnitřního stavu nebo na poslaný příkaz.

Pro každou sestavu řízenou Clarity existuje řídící modul ve formě dynamicky linkované knihovny. Před vypracováním bakalářské práce byl modul pro Chromaster téměř hotov a sestava Primaide vypadala podobně jako sestava Chromaster. Z toho vznikla myšlenka využít již provedenou práci na Chromasteru a vytvořit řídící modul pro sestavu Primaide.

Řídící modul pro Chromaster byl psán jako většina ostatních v jazyce C++ s použitím Clarity SDK. Z důvodu znovupoužitelnosti kódu, využití podobnosti a využití zkušeností s ostatními moduly psanými v C++ jsem u toho zůstal i pro Primaide.

1. Úvodní rozbor zadané problematiky

Funkční požadavky řídícího modulu

- detekovatelnost a fungování v aplikaci Clarity
- řízení soustavy Primaide
- chod v demo módu Clarity

Nefunkční požadavky řídícího modulu

- uživatelské prostředí v anglickém jazyce
- lokalizovatelnost do různých jazyků

1.1.Clarity

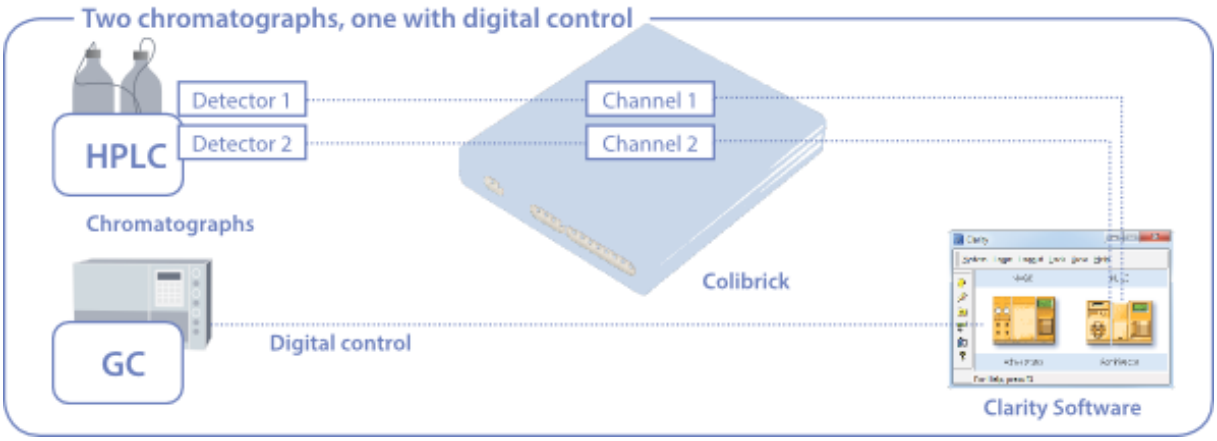
Clarity je chromatografická datová stanice (CDS) s volitelnými softwarovými moduly pro sběr a zpracování dat a řízení přístrojů. Její škála rozhraní pro sběr dat (A/D převodníky, LAN, USB, RS232) umožňuje připojení k prakticky kterémukoli chromatografu.

Měření

Až 4 nezávislé chromatografické systémy mohou být připojeny zároveň, každý může sbírat až 32 signálů z 12 detektorů.

Připojení

Jde použít s jakýmkoli GC nebo LC. Dohromady s volitelnými řídícími moduly a rozšířeními poskytuje laboratořím kompletní systém pro zacházení s daty.



Obrázek 1, příklad konfigurace připojení

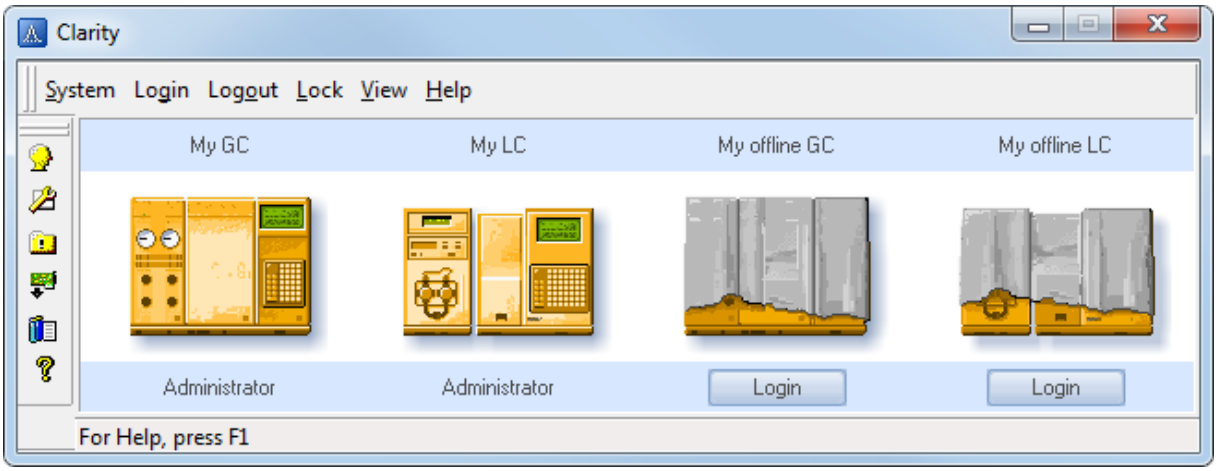
Příklad z obrázku 1 zobrazuje konfiguraci se dvěma chromatografickými systémy (GC + LC). GC je připojen a řízen přes digitální rozhraní a používá odpovídající řídící modul GC (Colibrick je USB dvoukanálový A/D převodník).

Řízení

Řídící moduly poskytují sjednocené řízení vybraných přístrojů, rozšíření poskytují funkce pro specifické oddělovací techniky jako PDA (photo-diode array) nebo GPC (gel permeation chromatography) analýzu. Běžící analýza může být kontrolována přes iPhone nebo mobil s Androidem přes aplikaci Clarity2Go.

Okna Clarity

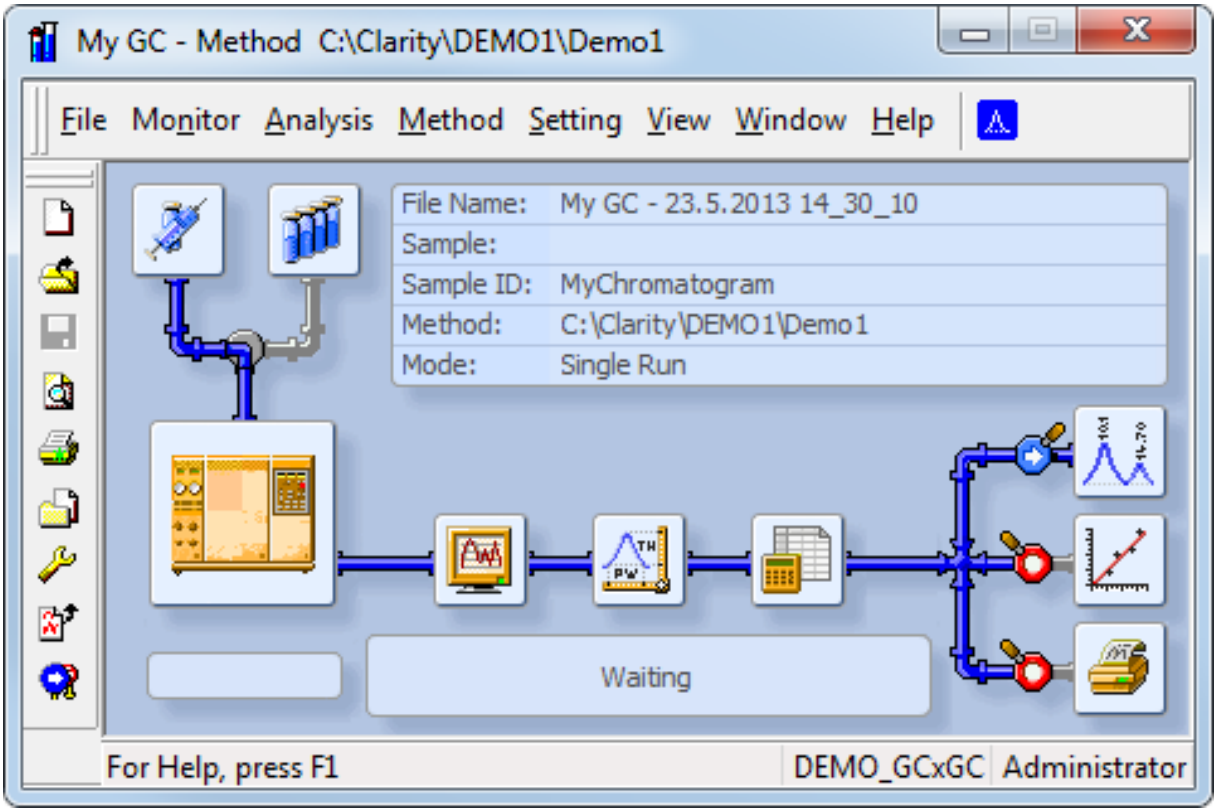
Clarity window



Obrázek 2, Clarity window

Okno Clarity je vstupním bodem do jednotlivých chromatografických systémů.

Instrument window

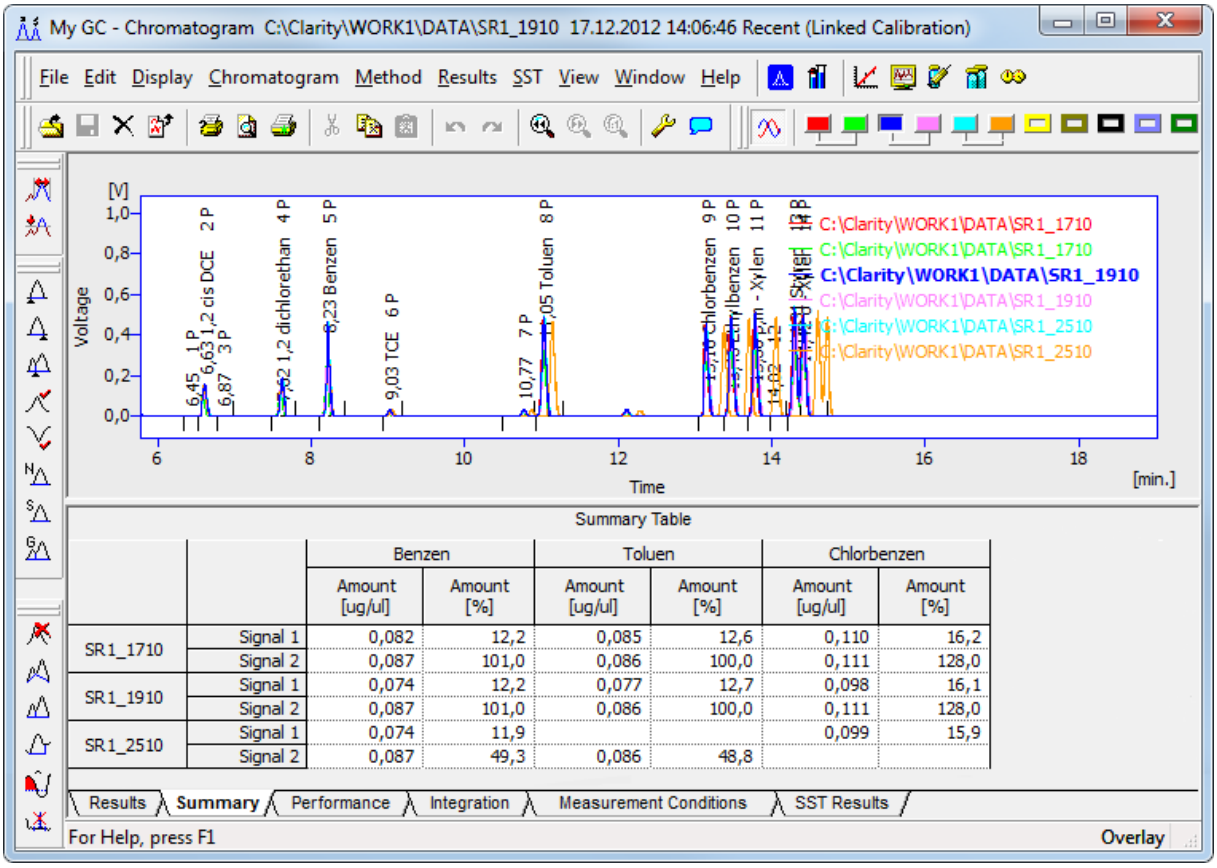


Obrázek 3, Instrument window

Okno přístroje je řídícím centrem celého procesu sběru dat a jejich vyhodnocování. Zahrnuje informační tabulku zobrazující zpracovávaný vzorek: jméno, použitá šablona chromatografické metody, způsob sběru dat atd. Stavový řádek ukazuje aktuální stav

analýzy: uplynulý čas a status. Diagram procesu analýzy poskytuje ikony pro rychlý přístup ke každému procesu celého postupu.

Chromatogram window

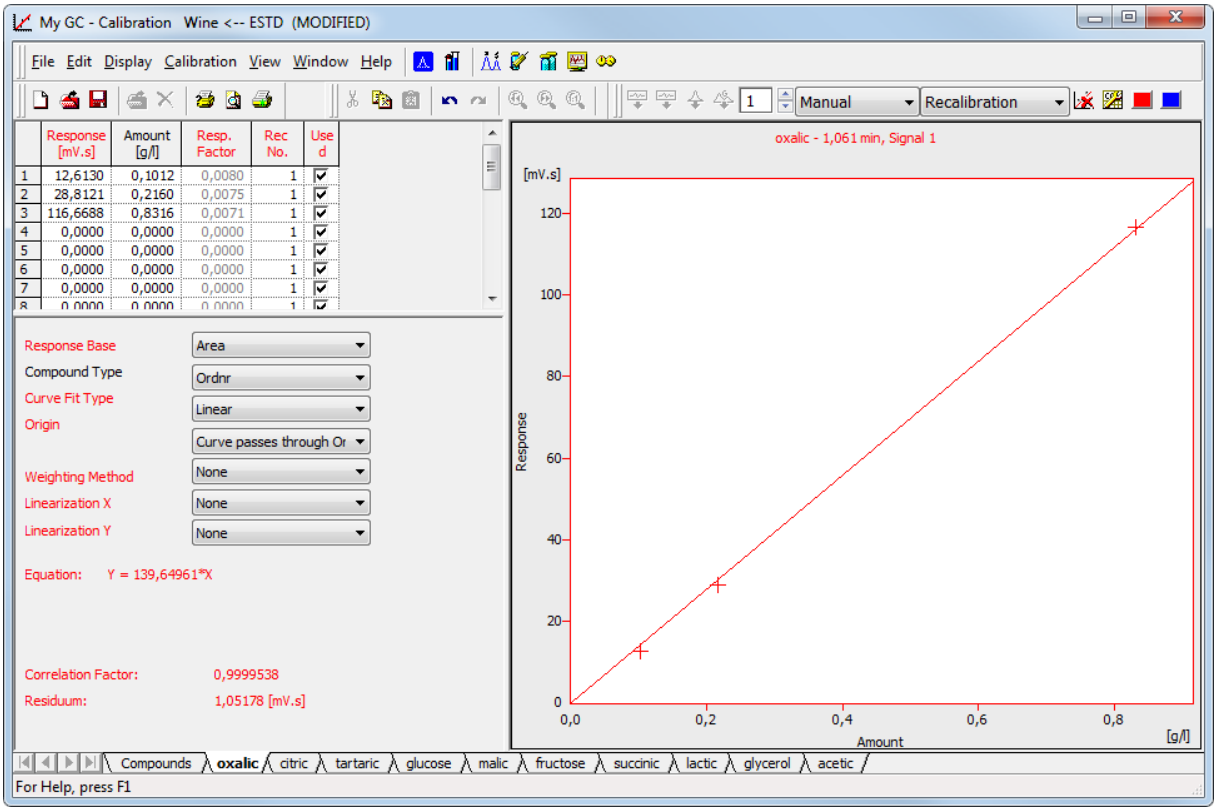


		Benzen		Toluen		Chlorbenzen	
		Amount [ug/ul]	Amount [%]	Amount [ug/ul]	Amount [%]	Amount [ug/ul]	Amount [%]
SR1_1710	Signal 1	0,082	12,2	0,085	12,6	0,110	16,2
	Signal 2	0,087	101,0	0,086	100,0	0,111	128,0
SR1_1910	Signal 1	0,074	12,2	0,077	12,7	0,098	16,1
	Signal 2	0,087	101,0	0,086	100,0	0,111	128,0
SR1_2510	Signal 1	0,074	11,9			0,099	15,9
	Signal 2	0,087	49,3	0,086	48,8		

Obrázek 4, Chromatogram window

Okno chromatogramu zobrazuje chromatogram a výsledky. Uživatelé mohou upravovat své chromatogramy v grafu nebo přes integrační tabulku.
Tabulky výsledků (Results, Summary, Performance atd.) mohou být přizpůsobeny k zobrazení dat, která chce uživatel vidět, způsobem, jakým je chce vidět.
Záložka Summary zobrazuje data z překrývajících se chromatogramů pro srovnání v jedné tabulce.

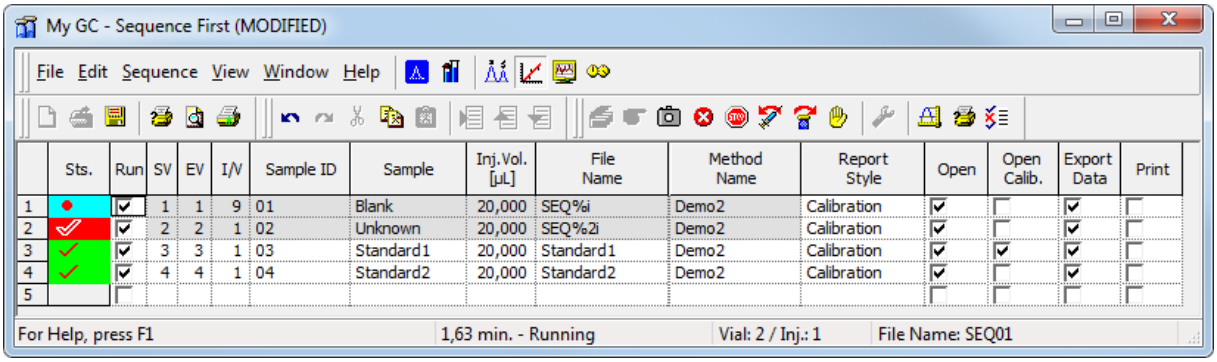
Calibration window



Obrázek 5, Calibration window

Okno kalibrace má 2 hlavní obrazovky: celkovou kalibrační tabulku a specializovanou záložku pro každou sloučeninu.
Záložka sloučeniny obsahuje tabulku s množstvím a úrovněmi koncentrace odpovídající odezvě a kalibrační křivku.

Sequence window



Obrázek 6, Sequence window

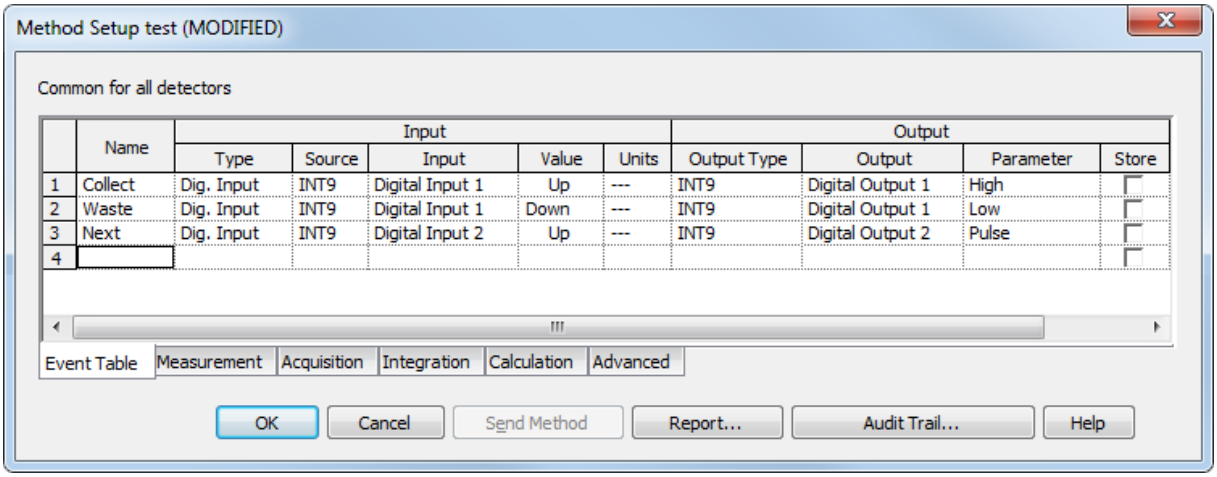
Sekvenční tabulka jde snadno editovat a stav každého řádku je ukázán barevnými symboly.

Device Monitor window

Obrázek 7, Device Monitor window

Monitor zařízení umožňuje řídit parametry přístrojů během analýzy.

Event Table



Obrázek 8, Event Table

Tabulka událostí umožňuje řízení vstupů a výstupů na základě událostí jako je změna stavu digitálního vstupu, úroveň signálu, čas analýzy atd. A/D převodníky poskytují digitální výstupy, které mohou být použity pro začátek synchronizace s dalšími přístroji. Tyto výstupy mohou být také řízeny z tabulky událostí.

Požadovaná PC konfigurace pro Clarity

	minimální	**doporučená**
Windows 8 (32bit, 64bit)	stejná jako pro MS Windows 8	
Windows 7 (32bit, 64bit)	stejná jako pro MS Windows 7	
Windows Vista (32bit, 64bit)	stejná jako pro MS Windows Vista	
Windows XP	PC Pentium III/700 MHz, 256 MB RAM	PC Pentium 4/2 GHz, 512MB RAM
Monitor	rozlišení 1024x768, 64K barev	rozlišení 1280x1024 nebo 1680x1050, 64K barev

1.2.Clarity SDK

Clarity může být konfigurována a řízena beze změny svého zdrojového kódu. Aby to bylo umožněno, chromatografický hardware musí být zastoupen softwarovým plug-inem komunikujícím s Clarity. Tento plug-in se nazývá řídící modul. Clarity také poskytuje sadu standartní funkcionality, oken, tabulek, výpočtů atd. Tato standartní funkcionalita může být rozšířena externím softwarovým plug-inem nazvaným rozšiřující modul.

Clarity SDK umožňuje tvorbu 3 typů softwarových projektů:

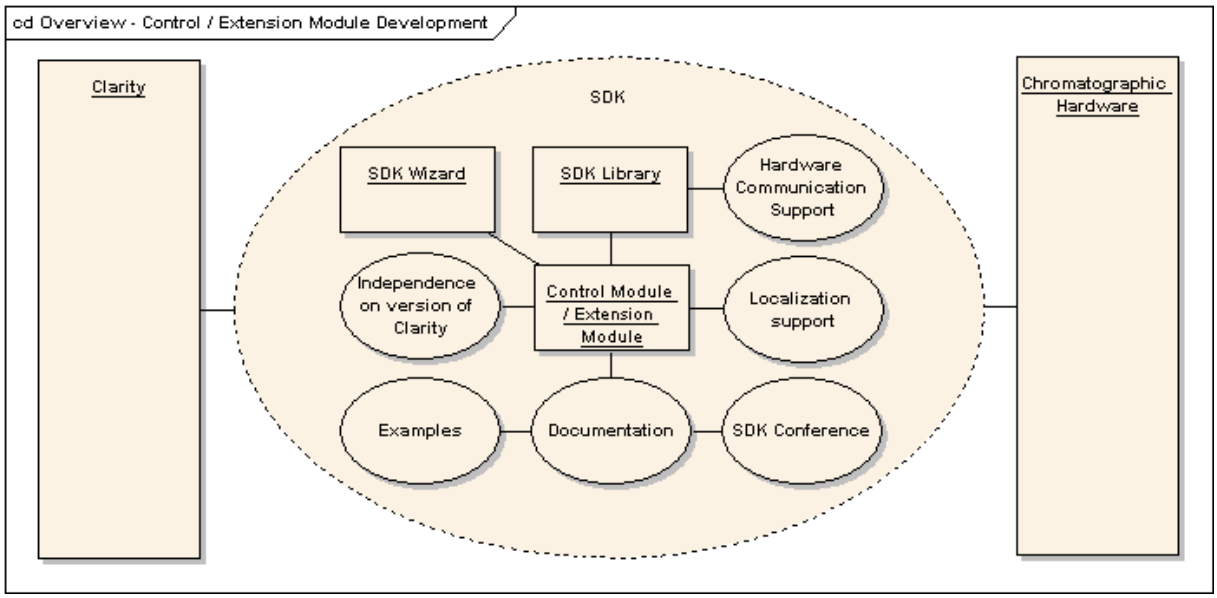
- řídící modul
- rozšiřující modul
- rozšiřující řídící modul

a poskytuje stabilní dokumentované rozhraní mezi Clarity a řídícím modulem.

Řídící modul překládá požadavky Clarity do jazyka chromatografického hardwaru a zpět překládá data získaná z hardwaru a synchronizuje události z jazyka hardwaru do volání funkcí, které Clarity očekává.

SDK také odstiňuje nízkoúrovňové implementační detaily COM rozhraní od programátora, poskytuje podporu pro běžné a opakující se úkoly vývoje řídícího modulu a nabízí podporu pro komunikaci s hardwarem. SDK je implementováno v C++ a používá MS MFC a knihovny ATL. Je navrženo, aby šlo použít s Microsoft Visual C++.

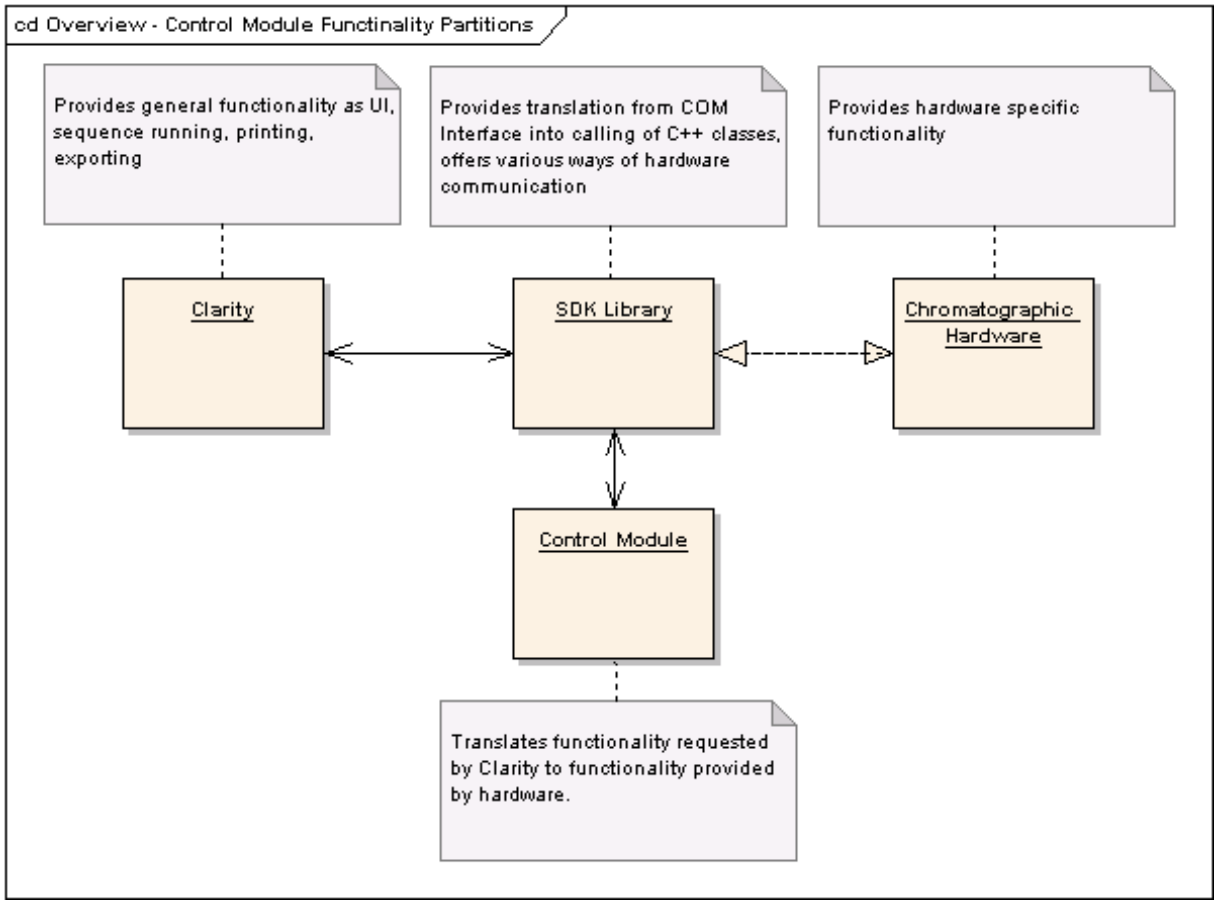
SDK se skládá z dynamicky linkované knihovny (CswSDKxx.DLL se svými zdroji), sady nástrojů (SDK Wizard, MSVC++ plugin, QTLinguist atd.) a dokumentace (CHM nápověda, příklady, SDK konference).



Obrázek 9, vývoj řídícího/rozšiřujícího modulu

Popis řídícího modulu

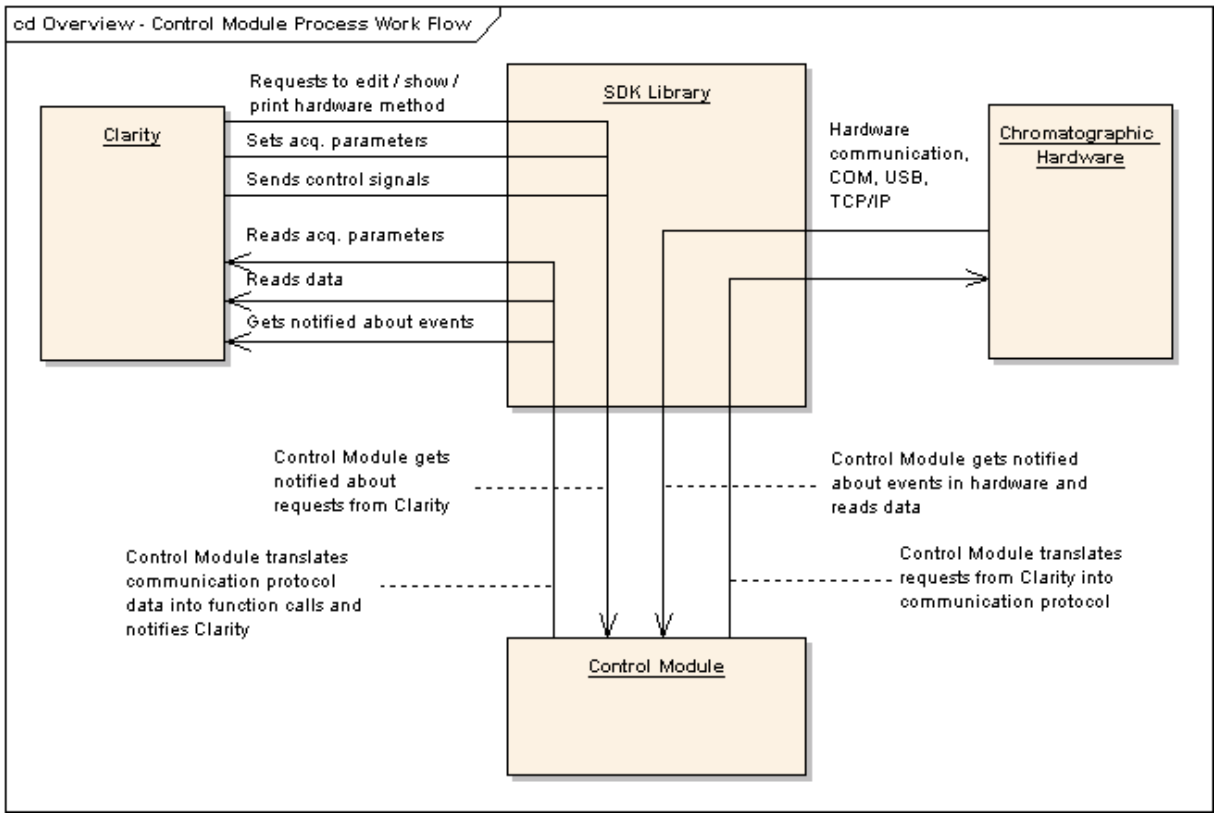
Struktura řídícího modulu



Obrázek 10, rozdělení funkcionality řídícího modulu

Základní operace řídícího modulu

Řídící modul obvykle zpracovává 3 věci, které by měly být sděleny Clarity: konfigurace přístroje (chromatografická metoda), sběr dat, synchronizace (např start, stop, ready atd.).

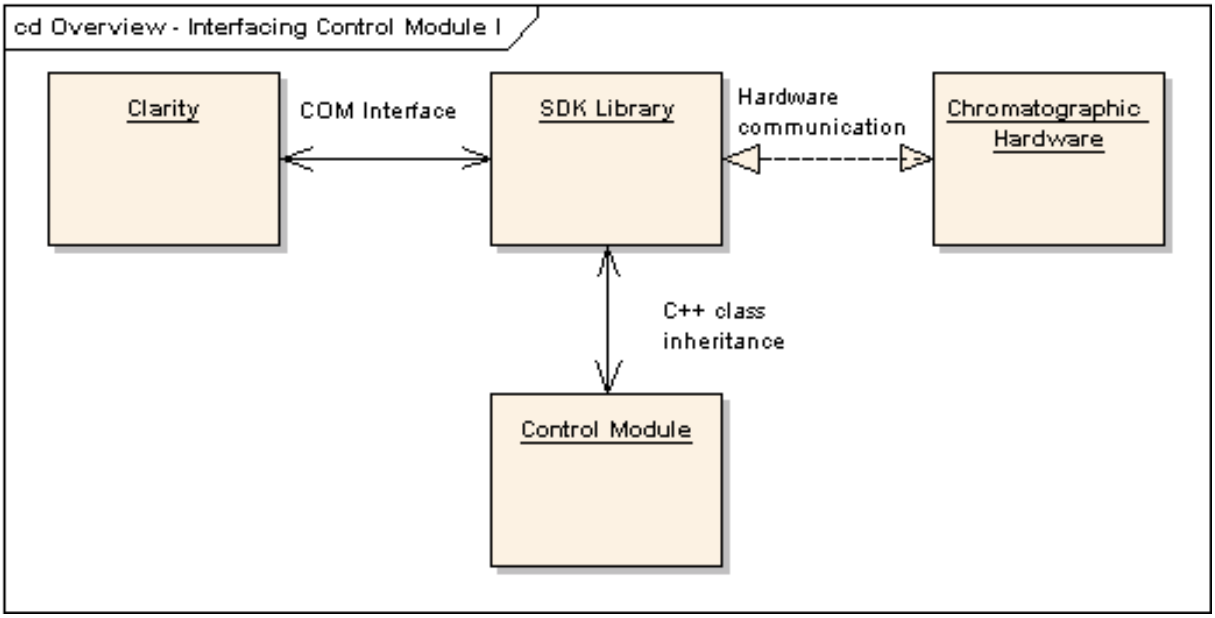


Obrázek 11, procesní workflow řídícího modulu

Obrázek 11 zobrazuje typický workflow zmíněných procesů mezi Clarity, Clarity SDK, řídícím modulem a hardwarem.

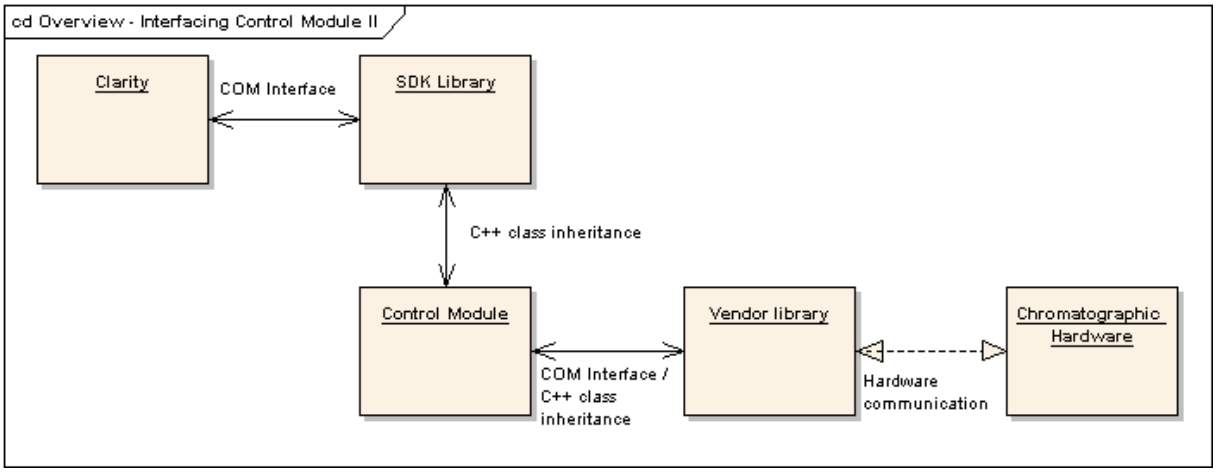
SDK staví na myšlence, že Clarity a části SDK volají metody SDK tříd. Některé z těchto metod jsou přepsány v řídícím modulu a dávají Clarity stručné rozhraní k hardwaru. Některé z těchto metod jsou abstraktní a musí se přepsat, ostatní mají výchozí implementaci, která se může měnit. SDK Wizard vytváří třídy a přepisuje metody, které se musí nebo by se měly implementovat pro hladký provoz. Téměř každá metoda v SDK třídách je virtuální, aby umožnila úplné ovládání přes chování řídícího modulu. Ze stejného důvodu přijímá většina metod nekonstantní argumenty i když je jejich výchozí implementace nemění.

Scénáře vývoje řídícího modulu



Obrázek 12, propojení řídícího modulu 1

Typický řídící modul komunikuje s hardwarem přes linku jako COM, USB, TCP/IP. V tomto scénáři může vývojář využít podporu hardwarové komunikace obsaženou v SDK. Vybere se odpovídající implementace komunikační linky (nazývaná pipe v SDK, vždy potomek rozhraní IPipe), např. CSerialPipe, CFTD2XXPipe atd. v SDK Wizardu a zavolá se metoda Send() v CSubdeviceProtocol::Cmd*().

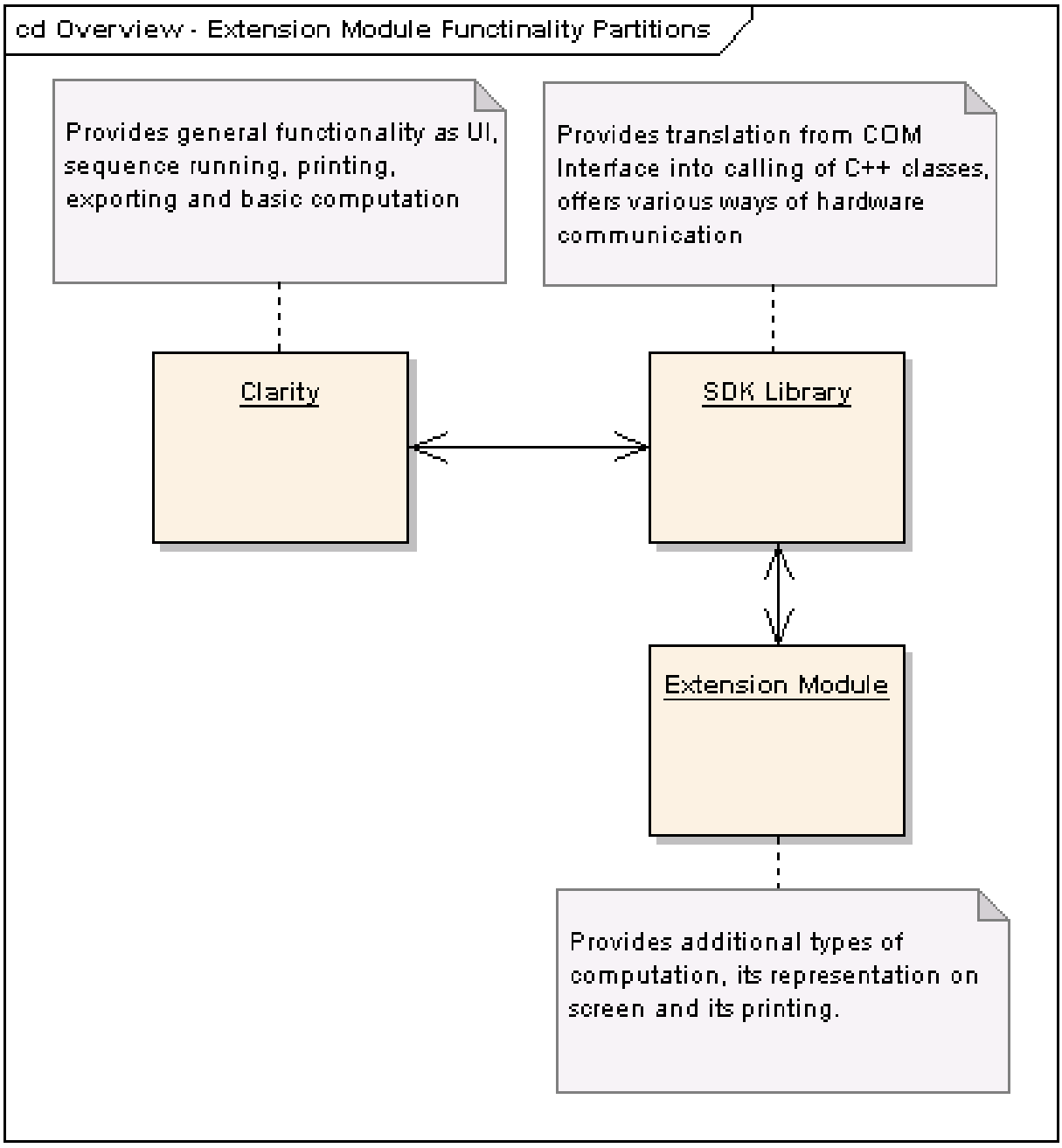


Obrázek 13, propojení řídícího modulu 2

Mnoho výrobců hardwaru začalo vyrábět vlastní SDK pro svůj hardware (COM objekty, DLL atd.) odstiňující hardwarovou komunikaci voláními funkcí na vyšší vrstvě. SDK také podporuje tento způsob vývoje řídících modulů, ale není to hlavním cílem. V SDK Wizardu se vybere CNullPipe jako implementace pipe a místo posílání dat přes pipe se volají funkce knihovny v metodě CSubdevice::Cmd*().

Popis rozšiřujícího modulu

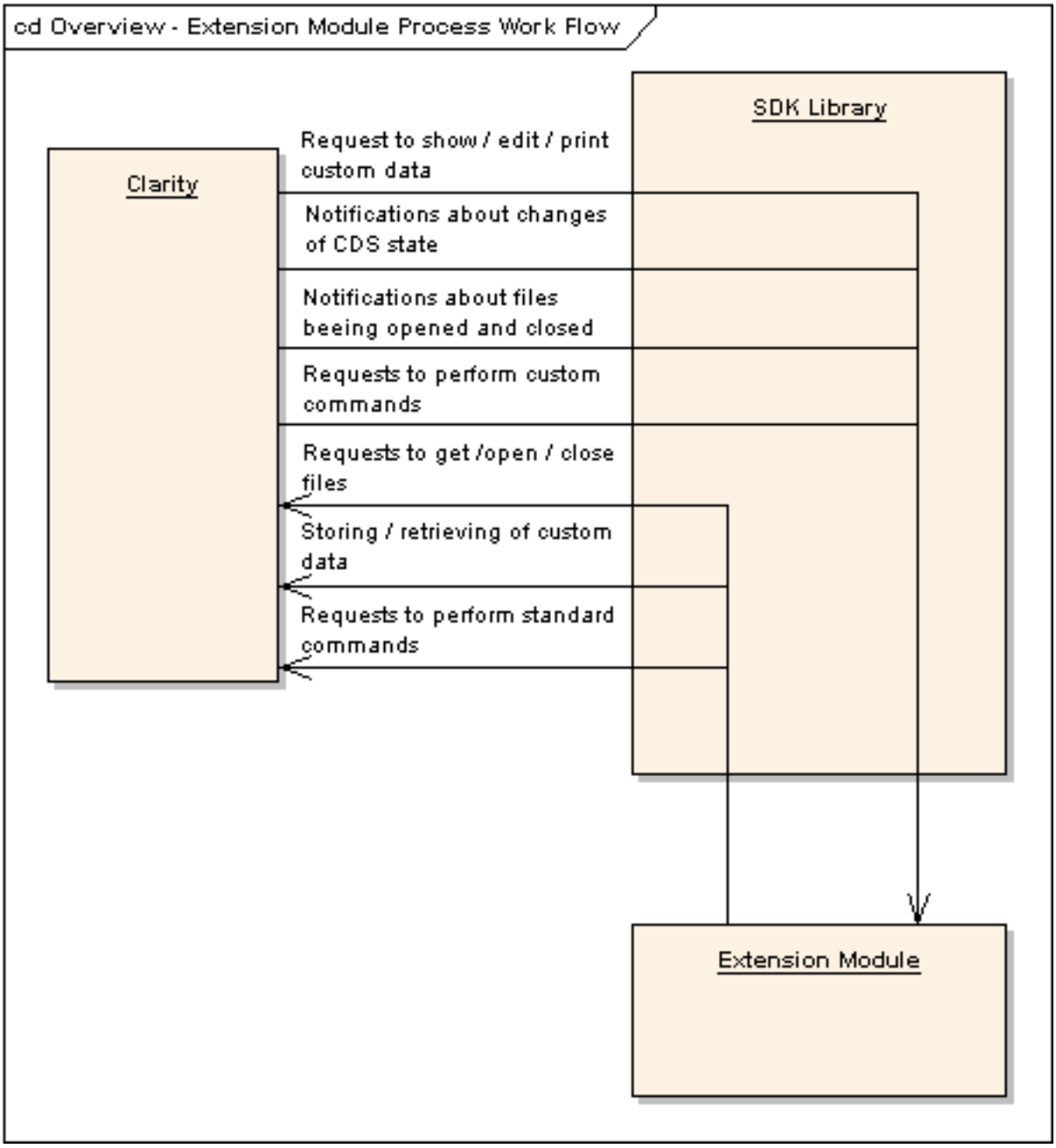
Struktura rozšiřujícího modulu



Obrázek 14, rozdělení funkcionality rozšiřujícího modulu

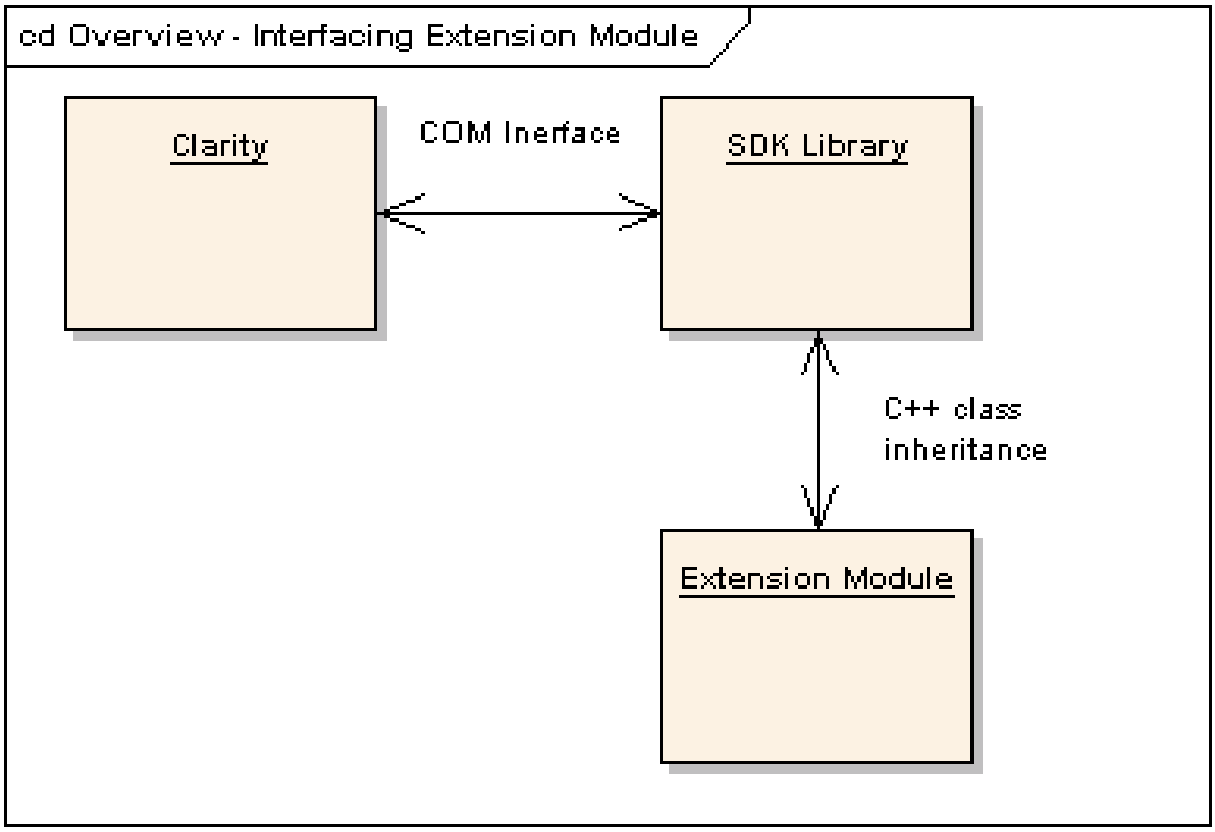
Základní operace rozšiřujícího modulu

Rozšiřující modul slouží jako vylepšení funkčnosti. Umožňuje provádění vlastních příkazů, zobrazení/úpravu/tisk vlastních dat a jejich uložení do jakéhokoli typu souboru podporovaného SDK. Také umožňuje provádění standartních příkazů Clarity.



Obrázek 15, procesní workflow rozšiřujícího modulu

Scénář vývoje rozšiřujícího modulu



Obrázek 16, propojení rozšiřujícího modulu

Popis rozšiřujícího řídícího modulu

Rozšiřující řídící modul obsahuje funkcionalitu jak řídícího tak rozšiřujícího modulu. To umožňuje vyvíjet řídící moduly se schopností číst data přímo z dříve naměřených chromatogramů nebo uchovat vlastní výsledky vzniklé během měření.

Požadavky pro vývoj SDK řídícího modulu

- Microsoft Windows XP nebo novější
- Microsoft Visual Studio 2010 (novější Visual Studia ještě nejsou podporovány; mohou se používat, ale je potřeba použít MFC verze 10). Expresní edice Visual Studia nemohou být použity, protože neobsahují MFC.
- aktuální verze stanice Clarity s USB hardwarovým klíčem

1.3.Hitachi Chromaster

Softwarová architektura

Aplikační software (řídící modul) komunikuje pomocí COM rozhraní s jednotlivými knihovnami modulů přístroje a ty komunikují přes USB se svým hardwarem.

Díky COM rozhraní je řízení přístroje z hlediska Clarity zcela transparentní a pokud podpůrný SW pracuje tak, jak má, je jeho vnitřní architektura pro Clarity vcelku nezajímavá.

Seznam modulů přístroje

Označení	Typ stroje	Soubor dokumentace
5110	Pump	CMDKIT5110Driver_R01.pdf
5210	Autosampler	CMDKIT5210Driver_R01.pdf
5310	Oven	CMDKIT5310Driver_R00.pdf
5410	UV Detector	CMDKIT5410Driver_R01.pdf
5420	UV-VIS Detector	CMDKIT5420Driver_R00.pdf
5430	DAD	CMDKIT5430Driver_R00.pdf
5440	FL Detector	CMDKIT5440Driver_R01.pdf
5450	RI Detector	CMDKIT5450Driver_R00.pdf

Problémy a zpoždění projektu

Při vývoji řídícího modulu pro Chromaster vznikly různé problémy. Nedařilo se spustit časový program, externí start soustavy nefungoval spolehlivě a nebylo zřejmé, jak se má u RI detektoru přepočítávat signál na jednotky RIU.

Projekt začal mít i časový skluz. Nejvýrazněji se to projevilo u detektorů, chybělo řízení k detektorům 5410 UV, 5440 FL a 5450 RI.

K projektu jsem byl přiřazen a implementoval jsem řízení chybějících detektorů. Odstranění chyb a vývoj chybějících detektorů je popsán v části „Popis řešení zadaného úkolu“.

Rozdíly mezi detektory

Narozdíl od ostatních modulů přístroje, všech 5 detektorů má většinu metod stejných, ale v ostatních metodách se lišily, následuje porovnání lišících se.

Metody pro práci s lampami

5410 UV Detector	5420 UV-VIS Detector	5430 DAD	5440 FL Detector	5450 RI Detector
bool Lamp_On()	bool D2Lamp_On()	bool D2_Lamp_On()	bool Lamp_On()	-
bool Lamp_Off()	bool D2Lamp_Off()	bool D2_Lamp_Off()	bool Lamp_Off()	-
-	bool WLamp_On()	bool W_Lamp_On()	-	-
-	bool WLamp_Off()	bool W_Lamp_Off()	-	-
-	bool Setup_LampMode(Mode<1-3>)	bool Setup_LampMode(Mode<1-3>)	-	-

Detektory UV, UV-VIS, DAD a FL mají na ověřování vlnové délky lampu Hg, která je zapnutá pořád. UV má navíc lampu D2, UV-VIS a DAD lampy D2 a W, FL lampu Xe. Detektory se dvěmi vypínatelnými lampami navíc nastavují mód pro provádění Auto Zero.

Metody pro nastavení

5410 UV Detector	5420 UV-VIS Detector	5430 DAD	5440 FL Detector	5450 RI Detector
bool Setup_UV_Param(..) stejně jako 5420	bool Setup_UVVIS_Param(..) stejně jako 5410	bool Setup_DAD_Param(Slit<1-2>) jiné parametry	bool Setup_FL_Param(Timeconstant<1-7>,Offset<0-1000>)	bool Setup_RI_Param(..) jiné parametry
bool Setup_RecorderRange(Range<1-2500>)	boolSetup_RecorderRange(Range<1-2500>)	-	bool Setup_RecorderRange(Range<1-1000>)	-
bool MoveWL(WL<190-900>)	bool MoveWL(WL<190-900>)	bool Setup_AnalogOutput(AbsRange<1-4>,WL1<190-900>,WL2<190-900>)	bool MoveWL(ExWL<0,200-850>,EmWL<0,250-900>)	-
bool Setup_AcqParameter(StopTime<1-36600>,Sampling<3-9>)	bool Setup_AcqParameter(StopTime<1-36600>,Sampling<3-9>)	bool Setup_Acq Parameter(..) víc parametrů	bool Setup_AcqParameter(StopTime<1-36600>,Sampling<3-9>)	boolSetup_AcqParameter(StopTime<1-36600>,Sampling<1-7>)

Různá nastavení detektorů.

Metody 2WL

5410 UV Detector	5420 UV-VIS Detector	5430 DAD	5440 FL Detector	5450 RI Detector
bool MoveWL_2WL(..) jiné délky než 5420	bool MoveWL_2WL(..) jiné délky než 5410	-	-	-
bool AutoZero_2WL()	bool AutoZero_2WL()	-	-	-
bool Setup_AcqParameter_2WL(..) jiné nastavení než 5420	bool Setup_AcqParameter_2WL(..) parametr "mode" navíc a jiné nastavení než 5410	-	-	-
bool Start_2WL(Start_Type<1-2>)	bool Start_2WL(Start_Type<1-2>)	-	-	-
bool Get_ArrayData_2WL(..)stejně jako 5420	bool Get_ArrayData_2WL(..)stejně jako 5410	-	-	-

Detektory UV a UV-VIS dokáží měřit na 2 různých vlnových délkách zároveň.

Metody pro práci s časovým programem

5410 UV Detector	5420 UV-VIS Detector	5430 DAD	5440 FL Detector	5450 RI Detector
bool Setup_TimeProgram(..) podobně jako 5420	bool Setup_TimeProgram(..) podobně jako 5410	-	bool Setup_TimeProgram(..) jiné parametry	-
bool Start_TimeProgram()	bool Start_TimeProgram()	-	bool Start_TimeProgram()	-
bool Stop_TimeProgram()	bool Stop_TimeProgram()	-	bool Stop_TimeProgram()	-

Časový program mají pouze detektory UV, UV-VIS a FL.

Metody na vyčítání stavu

5410 UV Detector	5420 UV-VIS Detector	5430 DAD	5440 FL Detector	5450 RI Detector
bool Get_Status2(..) podobně jako 5420	bool Get_Status2(..) podobně jako 5410	bool Get_Status2(..) jiné parametry	bool Get_Status2(..) jiné parametry	bool Get_Status(..) jiné parametry

Z detektorů se vyčítá aktuální stav.

Metody LogInformation

5410 UV Detector	5420 UV-VIS Detector	5430 DAD	5440 FL Detector	5450 RI Detector
bool Get_LogInformation(..) míň stavů lamp nez 5420/5430	bool Get_LogInformation(..) stejně jako 5430	bool Get_LogInformation(..) stejně jako 5420	bool Get_LogInformation(..) jiné nastavení parametrů LogInfo nez 5420/5430	-
bool Reset_LogInformation(..) míň stavů lamp než 5420/5430	bool Reset_LogInformation(..) stejně jako 5430	bool Reset_LogInformation(..) stejně jako 5420	bool Reset_LogInformation(..) jiné nastavení parametrů LogNum než 5420/5430	-

Detektory si ukládají informace o svých lampách.

5430 DAD má navíc odlišnosti:

- Initialize - chybí parametr CHNo<1-2>
- GetArrayData - navíc parametr pSpecNum pro určení počtu spekter v poli dat

5440 FL Detector má navíc metody:

- bool Set_PMT(PMT<1-5>)bool Set_Em_Bandwidth(Slit<1-2>)
- bool Set_XeLamp_ExchangeStandard(ExchangeStandard<1-2500>)
- bool Get_XeLamp_ExchangeStandard(pExchangeStandard<1-2500>)
- bool Transport_Position()
- bool WL_Check(..)
- bool Set_WLCalibration_Value(Value<1-20>)
- bool Set_WLCalibration_Value_Ex(Value<1-20>)
- bool Raman_Spectrum(..)
- bool Excitation_Spectrum(..)
- bool Emission_Spectrum(..)
- bool Excitation_EnergySpectrum(..)
- bool Emission_EnergySpectrum(..)

5450 RI Detector má navíc metody:

- bool Purge_On()
- bool Purge_Off()
- bool Setup_Temp(SetTemp<0,30-50>)

1.4.Hitachi Primaide

Softwarová architektura

Stejná jako u Chromasteru, ale knihovny modulů přístroje mají 3 vrstvy místo jedné. U Chromasteru komunikují knihovny rovnou přes USB, u Primaide používají ještě spustitelný soubor a další knihovnu.

Seznam modulů přístroje

Označení	Typ stroje	Soubor dokumentace
1110	Pump	1110_Primaide_00.pdf
1210	Autosampler	1210_Primaide_01.pdf
1310	Column Oven	1310_Primaode_01.pdf
1410	UV Detector	1410_Primaide_00.pdf
1430	Diode Array Detector	1430_Primaide_00.pdf

UV-VIS, FL a RI detektor v sestavě narozdíl od Chromasteru nejsou.

Možná řešení vývoje řídícího modulu pro Primaide

1. Vytvořit nový řídící modul a vyvíjet ho s Chromasterem paralelně.
2. Vytvořit nový řídící modul a vybrat funkční části, které by bylo vhodné sdílet s Chromasterem, specifické části vyvíjet paralelně.
3. Sdílet s Chromasterem jeden společný řídící modul, specifické části řešit virtuálním mechanismem.
4. Doprogramovat stávající řídící modul Chromaster, aby podporoval i sestavu Primaide.

V každém případě je dobré využít co nejvíce stávajícího kódu z řídícího modulu Chromaster.

2. Popis řešení zadaného úkolu

2.1.Hitachi Chromaster

Časový program

Od počátku vývoje se nedařilo spustit časový program. Hitachi při své návštěvě v Praze 12.3.2014 zjistilo, že poslalo stroj, který měl v nějakém interním nastavení zablokováno provádění časového programu. Vyřešeno resetem do továrního nastavení.

Hitachi nedokázalo odpovědět, jak zabránit provádění programu, když ho uživatel provádět nechce. Možné řešení bylo poslání prázdného programu, ale podle dokumentace má mít aspoň 2 řádky. Vyřešeno posláním 2 řádků, jeden s nastavením parametrů pro celou délku běhu detektoru a druhý s nejmenším časovým odstupem (6 sekund podle dokumentace) a s nastavením všech ostatních parametrů na hodnotu -1 (není použito).

Externí start

Před vyvoláním externího startu potřebuje detektor dostat příkaz Start s hodnotou "Synchronous Acquire", tím se přepne do stavu "Waiting for start", pak externí start zahájí sběr dat čili detektor se přepne do stavu "Acquiring data". To detekuje řídící modul a na jednu sekundu zapne virtuální digitální signál „Chromaster“.

Tento mechanismus funguje pro externí start i pro sekvenci. V druhém případě dá startovací signál automatický vzorkovač při ukončení nástřiku.

Nedá se ale poznat, kdy externí signál přijde. Řídící modul to řeší tak, že pošle příkaz Start s hodnotou "Synchronous Acquire" po každém odeslání chromatografické metody a po každém dokončení sběru dat. Detektory jsou tedy prakticky trvale ve stavu "Waiting for start". Tím je vyřešena i sekvence.

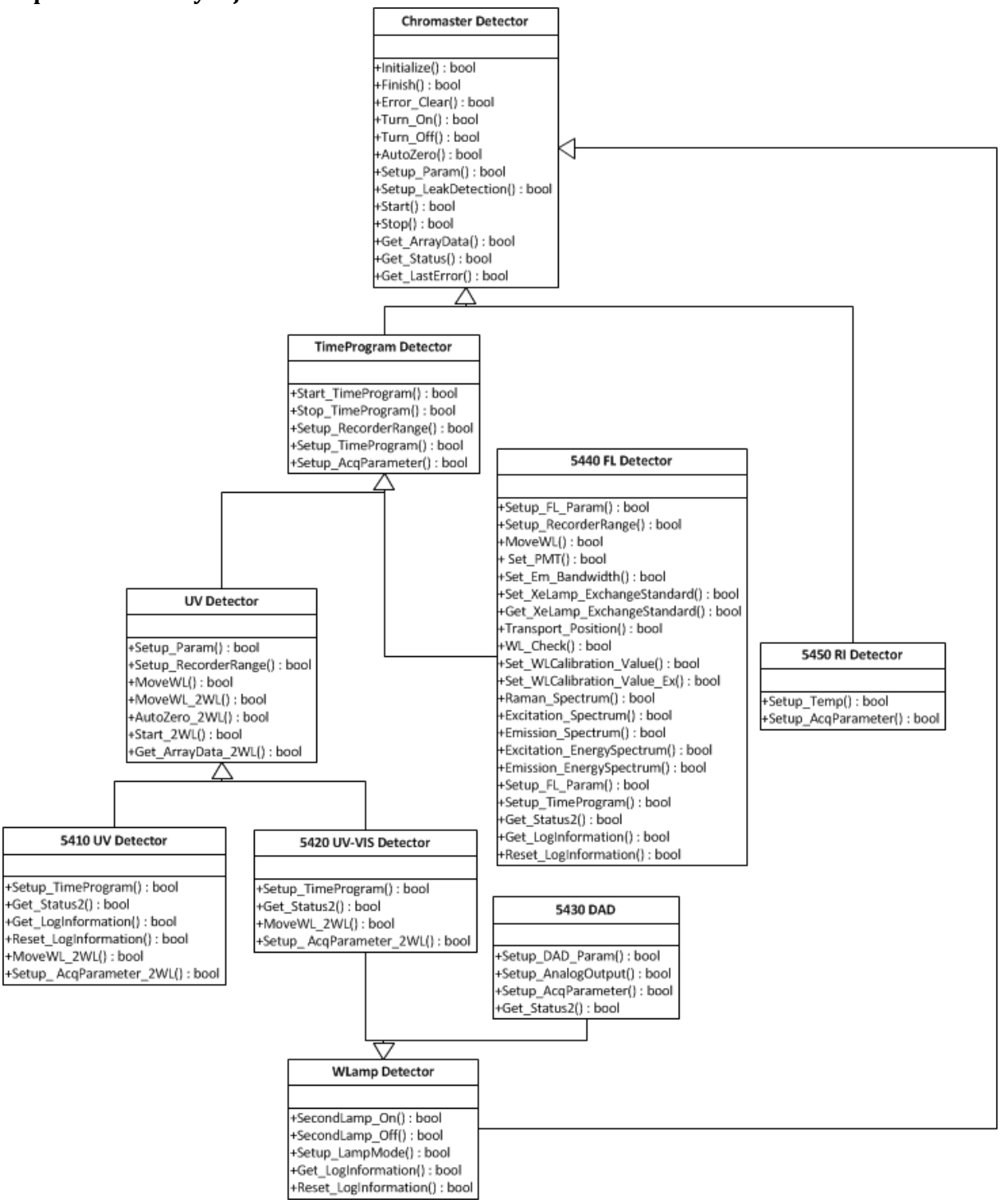
Přijde-li single run (jednoduché měření) z Clarity, řídící modul dá detektoru příkaz Stop a následně Start s hodnotou "Prompt Acquire".

UV a UV-Vis detektory v módu dvou vlnových délek reagují správně jen na první externí start. Pří dalším nepřejdou do stavu "Acquiring dual WL", jak by měly a příkaz Get_ArrayData_2WL trvale vrací nulový počet dat, což Clarity časem vyhodnotí jako komunikační chybu. Metodou pokus a omyl se zjistilo, že externí start funguje správně jen poprvé po odeslání metody do stroje, takže se mezi každým ukončením sběru dat a zavoláním příkazu Start_2WL s hodnotou "Synchronous Acquire" do detektoru odešle chromatografická metoda.

Přepočet signálu RI detektoru

Přepočet dat detektoru na µRIU je v dokumentaci poněkud zmatený. Podle mailu Hitachi 11.4.2014 byla určena převodní konstanta pro dělení z hodnoty signálu na hodnotu µRIU na 8000.

Implementace chybějícího řízení detektorů



Obrázek 17, hierarchie detektorů

Podle porovnání detektorů z části „Úvodní rozbor zadané problematiky“ jsem vytvořil diagram dědičnosti tříd protokolů detektorů přístroje Chromaster. Všechny Chromaster detektory dědí z šablonového (template) předka CChromaster_CM54X0DetectorProtocol.

CChromaster_CM54X0DetectorProtocol.h

```
/*direktivy pragma once a include, dopředné deklarace tříd*/
template<class _TChromasterDetectorClass, class _TDKITDriver>
// Common ancestor of all Chromaster detectors
class CChromaster_CM54X0DetectorProtocol : public CDetectorProtocol {
        typedef CDetectorProtocol base;
public:
        CChromaster_CM54X0DetectorProtocol(CSubDevice *pSubDevice);
/*deklarace metod*/
};
#include "CChromaster_CM54X0DetectorProtocol.inl"
```

CChromaster_CM54X0DetectorProtocol a jeho parametrizovaní potomci mají 2 šablonové parametry:

- _TChromasterDetectorClass – typ detektoru
- _TDKITDriver – Hitachi knihovna řídící detektor

Parametrizovaní potomci CChromaster_CM54X0DetectorProtocol jsou:

- CChromaster_CM54X0WLDetectorProtocol – pro detektory s dvěmi lampami
- CChromaster_CM54X0TPDetectorProtocol – pro detektory s časovým programem
- CChromaster_CM54X0UVDetectorProtocol – potomek protokolu výše, navíc obsahující hlavně „2WL“ metody

Zajímavá situace nastává u 5420 UV-VIS detektoru, protože tento neparametrický potomek třídy CChromaster_CM54X0DetectorProtocol dědí ze tříd CChromaster_CM54X0WLDetectorProtocol a CChromaster_CM54X0UVDetectorProtocol, tím vzniká situace známá jako problém diamantu a řeším jí virtuální dědičností.

```
template <class _TChromasterDetectorClass, class _TDKITDriver>
// Common ancestor of all Chromaster detectors with TimeProgram
class CChromaster_CM54X0TPDetectorProtocol : public virtual
CChromaster_CM54X0DetectorProtocol<_TChromasterDetectorClass, _TDKITDriver>
```

```
template <class _TChromasterDetectorClass, class _TDKITDriver>
// Common ancestor of both Chromaster detectors with second lamp
class CChromaster_CM54X0WLDetectorProtocol : public virtual
CChromaster_CM54X0DetectorProtocol<_TChromasterDetectorClass, _TDKITDriver>
```

CChromaster_CM5420UVVISDetectorProtocol.h

```
#pragma once
#pragma warning( disable : 4250 )// LUKAS - VS warning about correct behavior of overridden methods

#include "..\CChromaster_CM54X0UVDetectorProtocol.h"
#include "..\CChromaster_CM54X0WLDetectorProtocol.h"
#include "..\..\CMDKIT\CMDKIT5420.h"
#include "..\CChromaster_CM54X0DetectorProtocol.h"
#include "CChromaster_CM5420UVVISDetector.h"

class CChromaster_CM5420UVVISDetectorProtocol :
        public
CChromaster_CM54X0UVDetectorProtocol<CChromaster_CM5420UVVISDetector,
CCMDKIT5420>,
        public
CChromaster_CM54X0WLDetectorProtocol<CChromaster_CM5420UVVISDetector,
CCMDKIT5420> {
        typedef public
CChromaster_CM54X0WLDetectorProtocol<CChromaster_CM5420UVVISDetector,
CCMDKIT5420> base;
public:
        CChromaster_CM5420UVVISDetectorProtocol(CSubDevice *pSubDevice) :
CChromaster_CM54X0DetectorProtocol<CChromaster_CM5420UVVISDetector,
CCMDKIT5420>(pSubDevice),

CChromaster_CM54X0WLDetectorProtocol<CChromaster_CM5420UVVISDetector,
CCMDKIT5420>(pSubDevice),

CChromaster_CM54X0UVDetectorProtocol<CChromaster_CM5420UVVISDetector,
CCMDKIT5420>(pSubDevice) {};
/*deklarace metod*/
};
```

Je vidět odparametrizování třídy argumentem CChromaster_CM5420UVVISDetector za parametr _TChromasterDetectorClass a argumentem CCMDKIT5420 za parametr _TDKITDriver a konstruktor volající konstruktory tří nutných předků.

Stávající třídy protokolu byly převedeny na listy hierarchie dědičnosti a byly nově implementovány třídy dosud chybějících implementací detektorů.

2.2.Hitachi Primaide

Chtěl jsem použít co nejvíce hotového kódu z řídícího modulu pro Chromaster, protože většina funkcionality Primaide (bylo jí míň) už byla obsažena v Chromasteru. Z možných řešení vývoje řídícího modulu pro Primaide vypadá nejsnadněji a nejlépe:

4. Doprogramovat stávající řídící modul Chromaster, aby podporoval i sestavu Primaide.

Má to tyto výhody:

- nevytváření duplicitního kódu
- větší přehlednost (jedno umístění místo dvou a celková menší velikost kódu)
- pravděpodobná nižší časová náročnost než při vytváření dalšího řídícího modulu

Knihovny pro Chromaster a Primaide by měly moci existovat a být používány paralelně, proto je jednodušší doprogramovat je pomocí podmíněného překladu než určovat typ sestavy za běhu dynamicky.

Má to pravděpodobně i tyto výhody oproti dynamickému určování typu:

- ještě menší duplicita kódu
- ještě větší přehlednost
- ještě nižší časová náročnost

Nastavení překladu

V IDE Visual Studia, ve vlastnostech projektu CswHitachiChromaster je vytvořena kromě výchozí ještě další lokální překladová konfigurace. Existují 2 globální konfigurace, UDebug a URelease. Existují 4 lokální konfigurace projektu, UDebug, UDebug Primaide, URelease a URelease Primaide. Při zvolení globální konfigurace je nutno nastavit ještě lokální konfiguraci projektu CswHitachiChromaster přes BUILD->Configuration Manager..., buď verzi standartní (Chromaster) nebo Primaide.

Primaide verze UDebug i URelease mají nastaven příznak „PRIMAIDE“ a pomocné soubory i výsledné DLL mají jiné názvy, Chromaster je v nich nahrazen za Primaide.

Rozdíly mezi metodami sestavy Chromaster a Primaide

Nepoužité metody

Chromaster 5110 Pump	Primaide 1110 Pump
-	bool Key_Lock(Mode<0-1>)
bool Purge_On()	-
bool Purge_Off()	-
bool Open_Valve(Solvent<1-4>)	-
bool Setup_PurgeFLow(PurgeFlow<1-9999>)	-
bool Setup_MotorControl(MotorMode<1-2>, MotorFix<0,30-300>)	-
-	bool Get_System_Busy(pBusy<0-1>)
-	bool Set_System_Busy(pBusy<0-1>)
-	bool Start_Out()
bool Get_LogInformation(CHNo<1>,LogInfo<>)	bool Get_LogInformation(CHNo<1>,LogInfo<>)
bool Reset_LogInformation(CHNo<1>, LogNum<1-8>,Month<1-12>, Day<1-31>,Year<2000-2100>)	bool Reset_LogInformation(CHNo<1>, LogNum<1>,Month<1-12>, Day<1-31>,Year<2000-2100>)
bool Setup_Manual_Low(CHNo<1>, MinPress<0-611>, MaxPress<1-612>, ConcB<-1-1000>, ConcC<-1-1000>, ConcD<-1-1000>,Flow<-1-9999>)	bool Setup_Manual_Low(CHNo<1>, MinPress<-1-399>,MaxPress<-1,1-400>, ConcB<-1-0>,ConcC<-1-0>,ConcD<-1-0>, Flow<-1-9999>)
bool AutoPurge_On(Solvent<1-4>, PurgeFlow<1-9999>,Time<1-30>)	-
bool AutoPurge_Off()	-
bool PlungerWash_On(Wash_Time<0-300>)	-
bool PlungerWash_Off()	-
bool Setup_LowGRMode(Mode<1-2>)	-
bool AutoPurgeValve_Control(Position<0-1>)	-
bool Pause(Mode<0-1>)	-

Chromaster 5210 Autosampler	Primaide 1210 Autosampler
-	bool Key_Lock(Mode<0-1>)
-	bool Purge_Position()
bool Plunger_Wash(Wash_Time<1-999>)	bool Plunger_Wash(Wash_Volume<0.0-5000.0>)
bool Setup_Syringe(Syringe<1-3>)	bool Setup_Syringe(Syringe<1-5>)
-	bool Setup_RackParam(RackNo<1-16>,RX1<145-1390>, RX2<145-1390>,RXNum<1-40>, RY1<0-3075>,RY2<0-3075>, RYNum<1-40>,RZ<0-570>,RType<1-8>)
-	bool Start_Out()
bool Reset_LogInformation(CHNo<1>, LogNum<1-5>,Month<1-12>, Day<1-31>,Year<2000-2100>)	boolReset_LogInformation(CHNo<1>, LogNum<1-4>,Month<1-12>, Day<1-31>,Year<2000-2100>)
bool Needle_Wash(Wash_Solvent<1-2>, Wash_Time_Solvent1<1-999>, Wash_Time_Solvent2<1-999>)	-
bool RinsePort_Wash(Wash_Time<1-999>)	-
bool Syringe_Purge(SyringeStroke<1-20>, SyringeSpeed<1-5>)	-
bool Pump_Purge_On(Wash_Solvent<1-2>)	-
bool Pump_Purge_Off()	-

Chromaster 5310 Oven	Primaide 1310 Column Oven
bool Get_Valve_Information(pValve1<0-3>,PValve2<0>)	-
bool Get_Connect_Information(pValve1<0-3>,pValve2<0>, pColumnManagement<0-1>)	-
-	bool Key_Lock(Mode<0-1>)
bool Start_TimeProgram()	-
bool Stop_TimeProgram()	-
bool Setup_OutputTemp(DA_Output<1-2,99>)	-
-	bool Get_System_Busy(pBusy<0-1>)
-	bool Set_System_Busy(pBusy<0-1>)
-	bool Start_Out()

Chromaster 5410 UV Detector	Primaide 1410 UV Detector
-	bool Key_Lock(Mode<0-1>)
-	bool Setup_TimeProgUse(TimeProgMode<0-1>)
-	bool Get_System_Busy(pBusy<0-1>)
-	bool Set_System_Busy(pBusy<0-1>)
-	bool Start_Out()

Chromaster 5430 DAD	Primaide 1430 Diode Array Detector
-	bool Get_System_Busy(pBusy<0-1>)
-	bool Set_System_Busy(pBusy<0-1>)
-	bool Start_Out()

Pumpa

Chromaster 5110 Pump	Primaide 1110 Pump
bool Initialize(SystemID<1-4>,CHNo<1-3>, GR_Mode<1>,pProgNo[16],pSerialNo[10])	bool Initialize(SystemID<1-2>,CHNo<1-2>, GR_Mode<1>,pProgNo[16],pSerialNo[10])
podmíněný překlad disabluje v dialogu možnosti "USB Interface Board" Sys3 a Sys4, hodnota CHNo nebyla omezena, protože se v kódu používá jen s hodnotou 1	

Chromaster 5110 Pump	Primaide 1110 Pump
bool Setup_TimeProgram_Low(MinPress<0-611>, MaxPress<1-612>,StepNum<1-92>, Time<0-36000>,ConcB<-1-1000>,ConcC<-1-1000>, ConcD<-1-1000>,Flow<-1-9999>,Event1<-1-2>, Event2<-1-2>,Event3<-1-2>,Event4<-1-2>)	bool Setup_TimeProgram_Low(MinPress<0-399>, MaxPress<1-400>,StepNum<1-90>,Time<0-36000>, ConcB<-1-0>,ConcC<-1-0>,ConcD<-1-0>,Flow<-1-9999>, Event1<-1-2>,Event2<-1-2>,Event3<-1-2>,Event4<-1-2>)
podmíněný překlad nastavuje limitní hodnoty MinPress a MaxPress, StepNum byl 90 už u Chromasteru, ConcB/C/D je pro Primaide jen špatně popsána v dokumentaci	

Chromaster 5110 Pump	Primaide 1110 Pump
bool Get_Status2(CHNo<1>,pStatus<0-1>, pBusy_Status<0-7,9-11>, pError_Status<32-38,48-52,152>, pGather_Status<0-2>,pOn_Status<0-1>,pPress<>, pConcA<>,pConcB<>,pConcC<>,pConcD<>,pFlow<>, pValve1<1>,pValve2<1>,pEvent1<0-1>,pEvent2<0-1>, pEvent3<0-1>,pEvent4<0-1>,pPurgeValve<0-1>, pPurgeValve_Status<0-1>,pPWashPump<0-1>, pPWashPump_Status<0-1>,pDynamicMix<0-1>, pDegasser<0-1>,pDegasser_Status<0-1>, pAutoPurge_WaitTime<0.0-30.0>)	bool Get_Status(CHNo<1>,pStatus<0-1>, pBusy_Status<0-7,9-11>, pError_Status<32-38,42-45>, pGather_Status<0-2>,pKey_Lock<0-1>, pOn_Status<0-1>,pPress<>,pConcA<>, pConcB<>,pConcC<>,pConcD<>, pFlow<>,pValve1<-1,1-4>,pValve2<-1,1-4>, pEvent1<0-1>,pEvent2<0-1>,pEvent3<0-1>, pEvent4<0-1>)
KeyLock není použit, jinak všechny parametry pro Primaide už má i Chromaster a ty jsou shodné	

Automatický vzorkovač

Chromaster 5210 Autosampler	Primaide 1210 Autosampler
bool Initialize(SystemID<1-4>, pProgNo[16],pSerialNo[10])	bool Initialize(SystemID<1-2>, pProgNo[16],pSerialNo[10])
podmíněný překlad disabluje v dialogu možnosti "USB Interface Board" Sys3 a Sys4	

Chromaster 5210 Autosampler	Primaide 1210 Autosampler
bool Injection(RackNo<1-16>,VialNo<1-999>, Volume<0.0-690.0>,CycleTime<0.0-999.9>)	bool Injection(RackNo<1-16>,VialNo<1-999>, Volume<0.0-4950.0>,CycleTime<0.0-999.9>)
hodnota Volume nebyla omezena, protože je špatně popsána v dokumentaci	

Chromaster 5210 Autosampler	Primaide 1210 Autosampler
bool Setup_ThermoParam(Temp<1-45>, UseTolerance<0-1>,Tolerance<2-15>)	bool Setup_CoolingParam(CoolingTemp<1-35>, UseTolerance<0-1>,Tolerance<2-15>)
maximální hodnotu CoolingTemp upravuje podmíněný překlad	

Chromaster 5210 Autosampler	Primaide 1210 Autosampler
bool Setup_InjectionParam(InjMethod<1-3>, InjLeadVolume<0-3000>,InjRearVolume<0-3000>, InjFeedVolume<0-5000>,InjWasteVolume<0-6000>, InjAirVolume<0-200>,InjSyringeSpeedAsp<1-5>, InjSyringeSpeedDsp<1-5>,InjNeedleSpeed<1-2>, NeedleWashBeforeInj<0-1>,WashSolvent<1-2>, RinsePortWashTime<0-999>, NeedleWashTimeSolvent1<1-999>, NeedleWashTimeSolvent2<1-999>, IsPrungerWash<0-1>,PrungerWashTime<1-999>, IsVialDetect<0-1>)	bool Setup_InjectionParam(InjSyringeSpeed<1-5>, InjNeedleSpeed<1-2>,WashSyringeStroke<0-20>, WashSyringeSpeed<1-5>,NeedleWashTime<0-30>, IsPrungerWash<0-1>,PlungerWashVolume<0.0-5000.0>, IsVialDetect<0-1>)
nastavování parametrů upravuje podmíněný překlad	

Chromaster 5210 Autosampler	Primaide 1210 Autosampler
bool Get_Status2(pStatus<0-1>,pBusy_Status<0-8,10-13>, pError_Status<32-37,47-57,61-71,74,152>,pThermo<0-1>, pThermo_On<0-1>,pThermo_Temp<-10.0-100.0>, pPeltier_Temp<-10.0-100.0>,pDegasser<0-1>, pDegasser_Status<0-1>,pDoor_Status<0-1>, pEvent_Wait<0-1>)	bool Get_Status(pStatus<0-1>, pBusy_Status<0-8,10-13>, pError_Status<32-37,48-58,60-70,74>, pKey_Lock<0-1>,pCool<0-1>,pCool_On<0-1>, pCool_Temp<-10.0-100.0>)
KeyLock není použit, jinak všechny parametry pro Primaide už má i Chromaster a ty jsou shodné	

Chromaster 5210 Autosampler	Primaide 1210 Autosampler
-	bool Get_System_Busy(pBusy<0-1>)
-	bool Set_System_Busy(pBusy<0-1>)
podmíněný překlad přidává do kódu jedno volnání obou metod	

Pec

Chromaster 5310 Oven	Primaide 1310 Column Oven
bool Initialize(SystemID<1-4>, pProgNo[16],pSerialNo[10])	bool Initialize(SystemID<1-2>, pProgNo[16],pSerialNo[10])
podmíněný překlad disabluje v dialogu možnosti "USB Interface Board" Sys3 a Sys4	

Chromaster 5310 Oven	Primaide 1310 Column Oven
bool Valve1_Control(Valve_Pos<1-6>)	bool Valve_Control(Valve_Pos<1-3>)
podmíněný překlad disabluje v dialogu možnosti "Valve Position" 4, 5 a 6	

Chromaster 5310 Oven	Primaide 1310 Column Oven
bool Setup_Temp(Temp<1-85>,MAXTemp<20-95>, Tolerance<0.1-9.9>,WaitTime<0-99>)	bool Setup_Temp(Temp<1-65>,MAXTemp<6-70>, Tolerance<1-99>,WaitTime<0-99>)
podmíněný překlad nastavuje limitní hodnoty parametrů	

Chromaster 5310 Oven	Primaide 1310 Column Oven
bool Setup_TimeProgram(StepNum<1-92>,Time<0-6000>, Temp<-1,1-85>,Valve1<-1,1-6>)	-
podmíněně překládáno jen pro Chromaster	

Chromaster 5310 Oven	Primaide 1310 Column Oven
bool Setup_AcqParameter(StopTime<1-36600>)	bool Setup_AcqParameter(StopTime<1-36000>)
maximální StopTime byl už u Chromasteru 36000	

Chromaster 5310 Oven	Primaide 1310 Column Oven
bool Get_Status2(pStatus<0-1>, pBusy_Status<0-3,5-11>, pError_Status<32-38,41,48-54,56,60,152>, pGather_Status<0-2>,PValve_Status<0-6>, pWait_Time<0.0-99.0>,pOven_Temp<>, pAmbient_Temp<>,pColumn_Connect<0-1>)	bool Get_Status(pStatus<0-1>, pBusy_Status<0-3,5-10>, pError_Status<32-38,42-50>,pGather_Status<0-2>, pKey_Lock<0-1>,PValve_Status<0-3>, pWait_Time<0.0-99.0>,pOven_Temp<>, pAmbient_Temp<>)
KeyLock není použit, jinak všechny parametry pro Primaide už má i Chromaster a ty jsou shodné	

UV detektor

Chromaster 5410 UV Detector	Primaide 1410 UV Detector
bool Initialize(SystemID<1-4>, CHNo<1-2>,PProgNo[16],PSerialNo[10])	bool Initialize(SystemID<1-2>, CHNo<1-2>,PProgNo[16],PSerialNo[10])
podmíněný překlad disabluje v dialogu možnosti "USB Interface Board" Sys3 a Sys4	

Chromaster 5410 UV Detector	Primaide 1410 UV Detector
bool Setup_UV_Param(Timeconstant<1-7>, Offset<0-2000>,Processorrange<1-4>,Polarity<1-2>)	bool Setup_UV_Param(Timeconstant<1-7>, Offset<0-2000>,Processorrange<1-4>,Polarity<1-2>)
podmíněný překlad nastavuje jiné hodnoty Timeconstant	

Chromaster 5410 UV Detector	Primaide 1410 UV Detector
bool Setup_TimeProgram(StepNum<1-92>, Time<0-6000>,WL<-1,190-600>,Base<-1,1-2>)	bool Setup_TimeProgram(StepNum<1-20>, Time<0-6000>,WL<-1,190-600>,Base<-1,1-2>)
maximální StepNum byl už u Chromasteru 20	

Chromaster 5410 UV Detector	Primaide 1410 UV Detector
bool Setup_AcqParameter(StopTime<1-36600>,Sampling<3-9>)	bool Setup_AcqParameter(StopTime<1-36000>,Sampling<1-7>)
maximální StopTime byl už u Chromasteru 36000, podmíněný překlad nastavuje jiné hodnoty Sampling	

Chromaster 5410 UV Detector	Primaide 1410 UV Detector
bool Get_Status2(pStatus<0-1>, pBusy_Status<0-2,4-12,15-16>, pError_Status<16-19,21-22,32-38,48-49,52-55,66,152>, pGather_Status<0-2>,pLamp_Status<0-3>, pProg_Status<0-1>,pWL<>,pAbs_Data<>, pSampleEnergy<>,pRefEnergy <>,pWL2<>, pAbs_Data2<>,pSampleEnergy2<>, pRefEnergy2<>,pSetWLMode<0-1>, pThermoCell<0-1>,pThermoCell_On<0-1>, pThermoCell_Status<0-1>,pDAOption<0-1>)	bool Get_Status(pStatus<0-1>, pBusy_Status<0-2,4-12,15>, pError_Status<16-19,32-38,48-49,52-55>, pGather_Status<0-2>,pKey_Lock<0-1>, pLamp_Status<0-1>,pProg_Status<0-1>, pWL<>,pAbs_Data<>,pSampleEnergy<>,pRefEnergy<>)
upravuje podmíněný překlad	

Chromaster 5410 UV Detector	Primaide 1410 UV Detector
2WL metody	-
2WL metody podmíněně překládány jen pro Chromaster	

DAD

Chromaster 5430 DAD	Primaide 1430 Diode Array Detector
bool Initialize(SystemID<1-4>, pProgNo[16],pSerialNo[10])	bool Initialize(SystemID<1-2>, pProgNo[16],pSerialNo[10])
podmíněný překlad disabluje v dialogu možnosti "USB Interface Board" Sys3 a Sys4	

Chromaster 5430 DAD	Primaide 1430 Diode Array Detector
-	bool Lamp_On()
-	bool Lamp_Off()
bool D2_Lamp_On()	-
bool D2_Lamp_Off()	-
bool W_Lamp_On()	-
bool W_Lamp_Off()	-
podmíněný překlad mění vypínání a zapínání jednotlivých lamp na společné	

Chromaster 5430 DAD	Primaide 1430 Diode Array Detector
bool Setup_Lamp_Mode(LampMode<1-3>)	-
podmíněně překládáno jen pro Chromaster	

Chromaster 5430 DAD	Primaide 1430 Diode Array Detector
bool Setup_AcqParameter(StopTime<1-36600>, Sampling<3-9>,MinWL<190-850>, MaxWL<240-900>,Bandwidth<1-2,4,8,16>,Response<1-7>)	bool Setup_AcqParameter(StopTime<1-36000>, Sampling<1-7>,MinWL<190-850>, MaxWL<240-900>,Bandwidth<1-2,4,8,16>, Response<1-5>)
maximální StopTime byl už u Chromasteru 36000, podmíněný překlad nastavuje jiné hodnoty Sampling a Response	

Chromaster 5430 DAD	Primaide 1430 Diode Array Detector
bool Get_Status2(pStatus<0-1>, pBusy_Status<0,9,11-12>, pError_Status<32-37,48-50,52-56>, pGather_Status<0-2>,pD2_Lamp_Status<0-1>, pW_Lamp_Status<0-1>,pHg_Lamp_Status<0-1>, pSlit_Status<1-2>,pThermoCell<0-1>, pThermoCell_Use<0-1>,pThermoCell_Status<0-1>, pDAOption<0-1>,pError_Status_DTC<96-98,105,112>)	bool Get_Status(pStatus<0-1>, pBusy_Status<0,9,11-12>, pError_Status<32-37,48-50,52-56>, pGather_Status<0-1>,pKey_Lock<0>, pD2_Lamp_Status<0-1>,pW_Lamp_Status<0-1>, pHg_Lamp_Status<0-1>)
upravuje podmíněný překlad	

Ukázky kódu s podmíněným překladem

```
	if (pSpecific->m_bAutoZeroAtStart &&
#ifdef PRIMAIDE
		!pDriver->bAutoZero(&strErrMsg)
#else
		(pSubDeviceRWConfig->m_nWLMode == 0 ? !pDriver->bAutoZero(&strErrMsg)
: !pDriver->bAutoZero2WL(&strErrMsg))
#endif
		) {
			ReportError(esStopSingle_StopSeq,
ERROR_DEVICE_COMMAND_NOT_ACCEPTED, 0,
				(pSubDeviceRWConfig->m_nWLMode == 0 ? _T("AutoZero: ") :
_T("AutoZero2WL: ")) + strErrMsg);
			return false;
	}
```

nebo

```
#ifdef PRIMAIDE
bRetflag = pDriver->bGadgetOnOff(false, &strErrMsg);
#else
bRetflag = _lamp == D2Lamp ? pDriver->bGadgetOnOff(false, &strErrMsg) :
   pDriver->bSecondLampOnOff(false, &strErrMsg);
#endif
```

nebo

```
            dMin = CTemperature::Convert(1, CTemperature::tu_C, eTempUnits);

#ifdef PRIMAIDE

            dMax = CTemperature::Convert(65, CTemperature::tu_C, eTempUnits);

#else

            dMax = CTemperature::Convert(85, CTemperature::tu_C, eTempUnits);

#endif
```

Další rozdíly mezi kódem pro Chromaster a Primaide

Měnily se define, řetězce, uuid a helpstringy, metody nesouvisející se soustavou, dialogy property page, tisky, makra, fotka soustavy, vytváření tříd pomocí maker atd., uvádím příklady z různých částí kódu:

Makro _DEVICE_CONTROL_NAME_

```
#ifdef PRIMAIDE
#define _DEVICE_CONTROL_NAME_  "Hitachi Primaide"
#else
#define _DEVICE_CONTROL_NAME_  "Hitachi Chromaster"
#endif
```

uuid Type Library

```
[
        version(1.0),
#ifdef PRIMAIDE
        uuid(/*uuid Primaide*/),
        helpstring("Primaide 1.0 Type Library")
#else
        uuid(/*uuid Chromaster*/),
        helpstring("Chromaster 1.0 Type Library")
#endif
]
```

CChromaster_CM5410UVDetector::GetCMSubdeviceName()

```
CString CChromaster_CM5410UVDetector::GetCMSubdeviceName() {
	CString strName;
#ifdef PRIMAIDE
	VERIFY(strName.LoadString(IDS_MODULE_NAME_PM5410UVDetector));
#else
	VERIFY(strName.LoadString(IDS_MODULE_NAME_CM5410UVDetector));
#endif
	return strName;
}
```

Členské proměnné CChromaster_CM5210SamplerMethod

```
#ifdef PRIMAIDE
	int m_nInjSyringeSpeed;		///< Syringe Speed at Injection
	int m_nWashSyringeStroke;	///< Syringe Stroke Number at Injection Wash
	int m_nWashSyringeSpeed;	///< Syringe Speed at Injection Wash
	int m_nNeedleWashTime;		///< Needle Wash Time at Injection Wash
	double m_dPlungerWashVolume;	///< Pump Plunger Wash Liquid Volume at ..
#else
	int	m_nInjMethod;			///< Injection Method
	int	m_nInjLeadVolume;		///< Lead Volume (Cut)
	int	m_nInjRearVolume;		///<  Rear Volume (Cut)
	int	m_nInjFeedVolume;		///< Feed Volume (All)
	int	m_nInjWasteVolume;		///< Waste Volume (Loop)
	int	m_nInjAirVolume;		///< Air Volume
	int	m_nInjSyringeSpeedAsp;		///< Aspiration Syringe Speed
	int	m_nInjSyringeSpeedDsp;		///< Dispiration Syringe Speed
	bool	m_bNeedleWashBeforeInj;		///< Needle Wash before Injection On/Off
	int	m_nWashingSolvent;		///< Select Washing Solvent
	int	m_nRinsePortWashTime;		///< Rinse Port Washing Time
	int	m_nNeedleWashTimeSolvent1;	///< Needle Washing Time (Solvent1)
	int	m_nNeedleWashTimeSolvent2;	///< Needle Washing Time (Solvent2)
	int	m_nPlungerWashTime;		///< Pump Plunger Wash Time at ..
#endif
```

MODUL_NAME

```
#ifdef PRIMAIDE
#define MODUL_NAME				L"Primaide"
#else
#define MODUL_NAME				L"Chromaster"
#endif
```

Změny dialogů

```
#ifdef PRIMAIDE//zneviditelnit Chromaster buttony a zmensit jeho group box
        const int nChromasterButtons[] = {IDC_IFBOARD_SYS3, IDC_IFBOARD_SYS4, 0};
        ShowDlgItems(this, nChromasterButtons, false);
        CRect r1;
        GetDlgItem(IDC_IFBOARD_SYS1)->GetWindowRect(&r1);
        ScreenToClient(r1);
        CRect r2;
        GetDlgItem(IDC_IFBOARD)->GetWindowRect(&r2);
        ScreenToClient(r2);
        GetDlgItem(IDC_IFBOARD)->MoveWindow(r2.left, r2.top, 2*(r1.left - r2.left) +
r1.Width(), r2.Height(), false);
#endif
```

Vytvoření hlavní třídy modulu přístroje se svou knihovnou

```
#pragma once

#ifdef PRIMAIDE
#import "Primaide\DKIT1110Driver.dll" no_namespace, named_guids
#else
#import "CMDKIT5110Driver.dll" no_namespace, named_guids
#endif
#include "..\resource.h"
#include "CMDKITBase.h"

#ifdef PRIMAIDE
class CCMDKIT5110 : public CCMDKITBase<P1110Driver, I1110Driver,
   IDS_MODULE_NAME_PM5110LC>
#else
class CCMDKIT5110 : public CCMDKITBase<CM5110, ICM5110,
   IDS_MODULE_NAME_CM5110LC>
#endif
{/*zbytek třídy*/
```

3. Závěrečné zhodnocení výsledků

Problémy v okamžiku odevzdávání práce

1. Primaide 1430 Diode Array Detector je rozbitý a je nutná asistence Hitachi.
2. Autodetekce (prvotní připojení) na Primaide trvá déle než na Chromasteru a možná bude vhodné najít lepší způsob, jak moduly přístroje detekovat.
3. Umístění prvků v dialozích není optimálně doladěno.

Budoucí vývoj

Nejdůležitější úpravou před případným uvedením na trh je oprava umístění prvků v dialozích. Tato úprava už neovlivní funkčnost a je jednoduchá. Z pohledu vývoje ji bylo vhodné odložit na závěr, protože jen zpříjemní ovládání řídícího modulu.

V kódu pro Chromaster/Primaide je možné provést velký refaktoring, který by kód dále zjednodušil a umožnil by v budoucnu snažší přidávání nové funkcionality a jednodušší opravy stávajícího kódu v případě nalezení chyb.

Zrychlit autodetekci soustavy Primaide by bylo přínosné, ale přidaná hodnota by byla malá a časové nároky pravděpodobně velké.

Závěr

Vytvořil jsem řídící modul pro soustavu Primaide, který je možné po zaregistrování ve Windows používat v Clarity. Při vývoji nastala neočekávaná událost a sice zpoždění vývoje řídícího modulu pro sestavu Chromaster, ze kterého jsem měl vycházet. K projektu jsem byl přiřazen a s použitím šablon (template) a dědičnosti jsem provedl implementaci chybějících detektorů. Tím jsem ztratil hodně času, ale získal všechny potřebné informace o sestavě Chromaster a řídícím modulu pro ní. Naopak příjemné bylo zjištění, že sestava Primaide a knihovny k ní jsou oproti Chromasteru jednodušší a že se její řídící modul dá vyvinout pomocí stávajícího kódu a podmíněného překladu.

Testování řídícího modulu pro Primaide probíhá a kromě jednoho problému, který je již vyřešen, vše kromě 1430 Diode Array detektoru funguje. Tento detektor fungoval zpočátku také, ale pravděpodobně vinou hardwarové chyby přestal fungovat. Na chybu hardware poukazuje i to, že program pro údržbu přímo od firmy Hitachi, s ním také nekomunikuje.

Seznam použité literatury

1. **DataApex**, CLARITY CHROMATOGRAPHY STATION, [Online] 2014
   http://dataapex.com/product.php?id=clarity-std-detail.php
2. **DataApex**, CLARITY - MAIN FEATURES, [Online] 2014
   http://dataapex.com/product.php?id=clarity-std-detail.php
3. **DataApex**, CLARITY CHROMATOGRAPHY SOFTWARE - SCREENSHOTS, [Online] 2013
   http://dataapex.com/product.php?id=clarity-std-screenshots.php
4. **DataApex**, TUTORIALS, [Online] 2009
   http://dataapex.com/detail.php?id=tutorials.php
5. **WikipediE**, Chromatografie, [Online] 2013
   http://cs.wikipedia.org/wiki/Chromatografie
6. **DataApex**, DataApex Clarity SDK, [Offline] 2013
   CD\dokumentace\ClaritySDK.chm
7. **DataApex**, Confluence – interní dokumentace, [Offline] 2014
   intranet společnosti DataApex

Seznam obrázků

Příloha – Obsah přiloženého CD

- Autorun.inf
- bakalarka.docx
- bakalarka.pdf
- Install.exe

složka „DLL ridicich modulu“

- cppunit_dll.dll
- cswbasemfc10.dll
- CswExt.dll
- CswHitachiChromaster.dll
- CswHitachiPrimaide.dll
- cswsdk10mfc10.dll
- register.bat

složka „dokumentace“

- 1110_Primaide_00.pdf
- 1210_Primaide_01.pdf
- 1310_Primaode_01.pdf
- 1410_Primaide_00.pdf
- 1430_Primaide_00.pdf
- ClaritySDK.chm
- CMDKIT5110Driver_R01.pdf
- CMDKIT5210Driver_R01.pdf
- CMDKIT5310Driver_R00.pdf
- CMDKIT5410Driver_R01.pdf
- CMDKIT5420Driver_R00.pdf
- CMDKIT5430Driver_R00.pdf
- CMDKIT5440Driver_R01.pdf
- CMDKIT5450Driver_R00.pdf

složka „navod ke Clarity“

- T001-Clarity-Introductory.exe
- T002-Clarity-Data-Acquisition-Window.exe
- T003-Clarity-Instrument-window.exe

složka „obrazky sestav“

- device_CM.bmp
- device_PM.bmp

Příloha – Instalace Clarity a přidání řídícího modulu

1. spustit soubor Install.exe a řídit se pokyny
2. z adresáře, kam se Clarity nainstalovala, smazat soubor CSWAliases.ini, jinak by v kroku 7 nešel najít Hitachi Chromaster
3. zkopírovat složku „DLL ridicich modulu“ na místo, odkud mohou DLL běžet
4. spustit ze složky „DLL ridicich modulu“ soubor register.bat
5. spustit Clarity
6. z okna Clarity přejít přes tlačítko nalevo nebo přes horní menu do okna Configuration
7. stisknout vlevo dole tlačítko Add... a najít řídící modul Chromaster nebo Primaide, nejlepší způsob je asi začít psát název a okno začne filtrovat řídící moduly
8. používat Clarity podle souborů ve složce „navod ke Clarity“